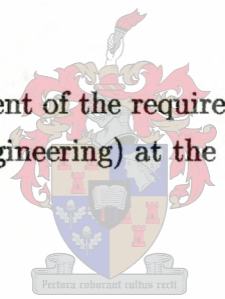
Structure from Motion estimation using a Nonlinear Kalman Filter

CHRIS VENTER

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic Engineering) at the University of Stellenbosch



Supervisor: Prof. B.M. Herbst
Co-supervisor: Prof. J.G. Lourens

December, 2002

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Date : November 25, 2002

Abstract

Structure from Motion is defined as the problem of extracting the 3d motion of a camera moving through a scene, as well as the 3d structure of the scene, from the image sequence produced by the camera over time. Several methods based on the Kalman filter have been proposed in the past, mostly based on the Extended Kalman filter. We propose an algorithm based on the dual Unscented Kalman filter to estimate the structure and motion of an object under perspective projection. It is shown that the algorithm is stable and accurate under synthetic as well as real-world conditions.

Opsomming

Struktuur vanuit Beweging is 'n rekenaar-visie probleem waarin die 3d beweging van 'n kamera deur 'n ruimte, asook die 3d struktuur van die ruimte, bepaal moet word slegs vanuit die 2d beelde in die beeldreeks wat deur die kamera geneem word. 'n Verskeie reeks oplossings, gebaseer op die Kalman filter, is reeds voorgestel om die probleem op te los. Meeste van die oplossings implementeer die "Extended Kalman filter", of EKF. Ons stel 'n algoritme voor, gebaseer op 'n nuwe nie-lineêre benadering tot die Kalman filter, die sogenaamde "Unscented Kalman filter", of UKF. Hierdie algoritme bepaal die struktuur en beweging onder 'n perspektief-projeksie kamera. Daar word getoon dat die algoritme stabiel en akkuraat funskioneer onder sintetiese sowel as reële toevoer.

Acknowledgements

Thanks goes out to my project leader, Professor Ben Herbst, whose energy and enthusiasm is matched only by his understanding of the subtle art of applied mathematics. Without him, the project would have been impossible to attempt.

Trizanne Pansegrouw, my belay-partner, dive-buddy, inspiration and guiding light: this project is dedicated to her.

Brian O'Kennedy, who listened to all the complaints I had when things didn't go as planned, which happened quite often.

The people of the DSP lab at the University of Stellenbosch, the residents of Farside and Parkhof, you all deserve my thanks.

The last, but not least, thanks goes out to my parents and other friends, and the people on the Internet that provided help (and code) without expecting anything in return.

to Trizanne

Contents

List of Figures

Notation

Note: any exceptions to the following notations will be specified where they occur.

Notation	*Description*
a or A	Scalar value
$\mathbf{a}$	Column vector
$\mathbf{a}^T$	The transpose of $\mathbf{a}$.
$\mathbf{A}$	A matrix
$F()$ or $F[]$	A function
$a(\tau_k)$	The value of a at time τ_k.
a_n	Refers to the nth component of $\mathbf{a}$.
$a^-(\tau_{k+1}\|\tau_k)$	The *a priori* value of a at time τ_{k+1}, based on its value up to time τ_k.
$\hat{a}$	Indicates that $\hat{a}$ is an *estimate* instead of the true value a
$a + b\mathbf{i}$	A complex number. a and b are scalars, with $\mathbf{i}$ the imaginary unit.
$a+b\mathbf{i}+c\mathbf{j}+d\mathbf{k}$	A quaternion. a is the scalar component, and (b, c, d) is the vector component.

Chapter 1

Introduction

The human visual system is mainly based on stereovision, where the 3d structure of the surroundings is resolved using the disparity between the two images produced by our eyes.

Although this ability is learnt immediately after birth, it is still the enormous processing power of the human brain which allows us to perceive depth in the way that we do. This becomes clear when we try to implement an algorithm to enable a computer to do the same. Steven Pinker writes: "...inverse optics, the deduction of an object's shape and substance from its [2 dimensional] projection, is an 'ill-posed problem'," in his book *How the mind works* [29].

However, a person who has use of only one eye still perceives depth, in some cases as well as if he had full use of two. Good examples of such people include "the aviator Wiley Post and a wide receiver for the New York Giants football team in the 1970's," says Pinker. This prompts us to ask the question: To what extent are we able to extract 3d information from the output of only one camera? "Keep one eye closed for a few minutes as you walk around," suggests Pinker, and notes that contrary to our expectation, the world does not lose depth. This because the mind has access to other types of information. "Most important," he notes, "it has motion".

This leads to the Structure from Motion problem, in which we must extract the the 3d motion of a camera, as well as the 3d structure of its surroundings, using only the 2d image sequence it produces.

It turns out that this type of solution is possible, however we are only ever able to reconstruct the 3d information up to an unknown scale factor, unless we have some kind of reference information. Imagine a boulder on the moon, with no trees, houses, or people next to it in order to provide some kind of reference. The boulder may appear to be a couple of feet tall and a hundred feet away, when it may in actual fact be several kilometres away and several hundred feet tall. Pinker gives another example of how the mind can be "tricked": it is found in the work of Adelbert Ames, Jr., who built an illusory room with slanted sides. The room appears to be a normal rectangular room, and the mind takes this into account when judging the size of an object in one of the corners. Therefore, an

child standing in the corner may be perceived as being larger than an onlooking adult.

The applications for SfM (Structure from Motion) solutions are numerous. In the field of robotics, the ability to extract 3d information from a scene enables an automaton, whether it is a self-driving car, or an articulated robot arm, to navigate and interact with its surroundings. In the entertainment industry, computer generated graphics must be integrated into existing footage of a real scene, in order to seduce the viewer into believing that it is a physical part of the scene. As such it is a commercially viable area of research.

In Computer Vision, the SfM problem has been approached from many different angles. Some proposed solutions attempt to extract the motion of the camera, independently from the structure of the scene, which may be extracted as a post-processing step. Others record a batch of frames, and use linear methods to extract the motion and structure from the object.

One specific area is perhaps more akin to the human visual system. Proposed solutions in this area process the sequence frame by frame, over time, as new measurements are made, in order to improve the estimate of the structure and motion of the system. The Kalman filter, which represents the estimate of a system with a Gaussian distribution, lends itself to such a recursive estimation framework.

The perspective camera SfM problem is however non-linear. Since non-linear transforms of Gaussian distributions do not usually produce Gaussian distributions, an approximation of the Kalman filter, called the Extended Kalman filter, has been used in the past. The EKF solves the non-linear propagation problem by transforming the covariance in a different linearised way than it updates the mean of the distribution.

In this thesis, we propose an algorithm which uses a newly developed non-linear Kalman filter approximation, called the Unscented Kalman filter, in order to solve the SfM problem. The algorithm compares favourably against an existing Kalman filter solution which implements the Extended Kalman filter. It is shown that the new filter delivers accurate results under real-world conditions even when it is provided with no initial information.

1.1 Overview of the document

Chapter 2 provides a view on the previous work done in the SfM field. It also introduces the perspective projection model, and explains why the SfM problem is a non-trivial one. Chapter 3 describes the Kalman filter as a state estimator, and provides an overview of a non-linear Kalman filter approximation, called the Extended Kalman filter. Thereafter it introduces the Unscented Kalman filter, which is another approximation of a non-linear Kalman filter. In chapter 4 the detailed implementation of the dual-UKF estimation models used in our algorithm is discussed. Chapter 5 displays and discusses the results

produced in a number of experiments. Chapter 6 summarises and concludes the work, and points out areas of possible improvement.

Chapter 2

The Structure from Motion problem

In general terms, the problem facing SfM is to calculate the 3d structure and motion of an object from the 2d projections it generates in a video sequence.

A general study of the information inherent in 3d image plane projections over time is conducted first. This is followed by a discussion of some of the different approaches found in the SfM field, starting with techniques used in stereovision and concluding with Kalman filtering methods. This will pave the way for an explanation of the proposed algorithm in the rest of the thesis.

2.1 The 2d projection of 3d motion

This section introduces the simple pinhole camera model, which is a good approximation of real-world lens systems found in cameras. The difficulty in extracting 3d information from single 2d images is explained first, followed by an explanation of why this is possible if a *sequence* of images is available.

2.1.1 A simple projection model

The simple pinhole camera used in this work is defined as having a focal point at $(0, 0, -f)$. It looks at the origin, with the positive x-axis pointing toward the right, and the positive y-axis pointing upward. The image plane coincides with the xy-plane. This coordinate system is called the *Camera coordinate system*, or CCS, and is illustrated in figure 2.1.

A point in the CCS will be defined as $\mathbf{p} = (p_x,\ p_y,\ p_z)$. The pinhole projection of this

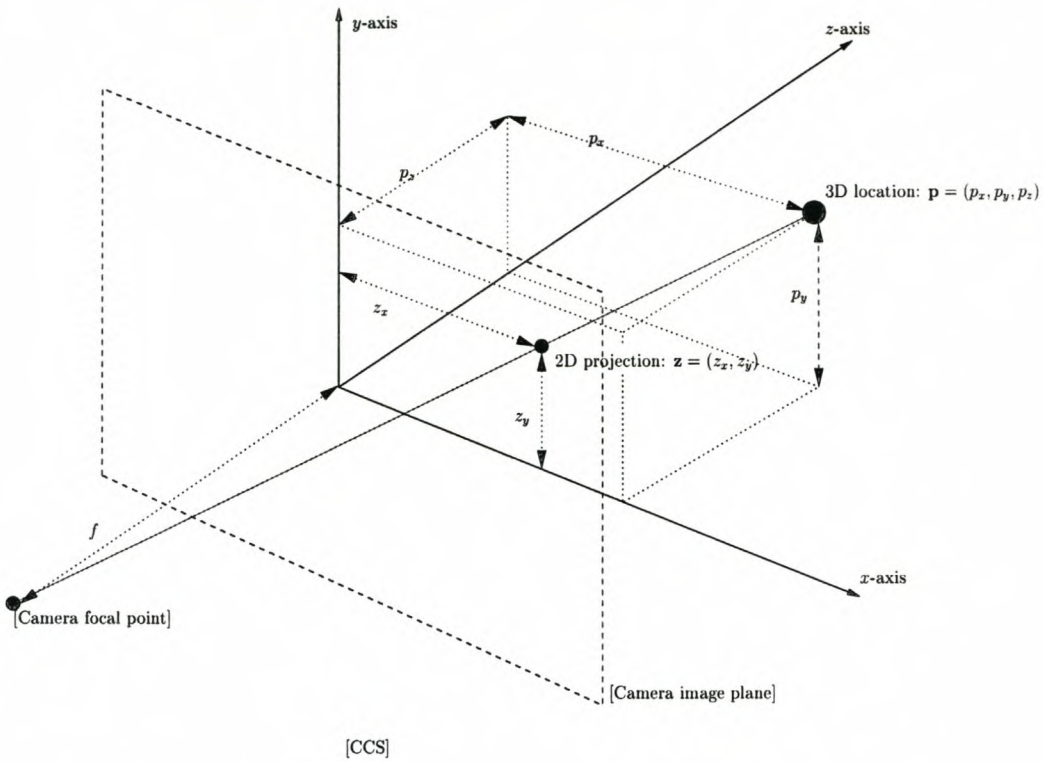


Figure 2.1: *The simple pinhole projection of a 3d point onto a 2d camera plane, located in the object-centred coordinate system.*

3d point onto the image plane of the pinhole camera is defined as

$$\begin{aligned} \mathbf{z} &= H[\mathbf{p}] && (2.1)\\ &= \frac{f}{p_z + f}\begin{bmatrix} p_x \\ p_y \end{bmatrix} && (2.2)\\ &= \frac{1}{1 + p_z/f}\begin{bmatrix} p_x \\ p_y \end{bmatrix}, && (2.3) \end{aligned}$$

where $\mathbf{z} = (z_x, z_y)$ is the image plane projection. $H[\cdot]$ is the non-linear projection function, in this case a *simple pinhole camera model.*

Note that when the focal length becomes infinite, the simple pinhole camera reduces to the orthographic camera, in which the projection of a 3d point is simply its perpendicular projection onto the xy-plane:

$$\lim_{f \to \infty} \frac{f}{p_z + f}\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \end{bmatrix}. \qquad (2.4)$$

2.1.2 A single 2d image

Consider a single 2d image from which we are able to extract an object's features. This information only provides us with z_x and z_y image plane coordinates for each point.

The goal of the structure part in SfM is to recover the 3d CCS coordinates of each point. If we rewrite (2.3) as

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} z_x(1 + p_z/f) \\ z_y(1 + p_z/f) \\ p_z \end{bmatrix}, \tag{2.5}$$

it is clear that we need an extra piece of information, in the form of the z-component of each point, in order to determine the 3d structure of the object. In the simple pinhole model, this information is "hidden" in the 2d information during projection. In the orthographic model, this information is simply discarded during projection. It is therefore not possible to recover the required extra information if we are only supplied with the 2d image plane projections from a single image.

The goal of the motion part in SfM is to recover the 3d CCS motion of the object. Since the object is rigid, this motion is shared by each point. Once again, if we only have access to a single frame's 2d image plane projections, there is no way of determining motion.

Note that there exists research in the area of Shape from Shading, which attempts to extract 3d structure information from a single image. However these algorithms use knowledge of the way a light source interacts with object surfaces, as well as segments from the image, to infer structure from a single image. Examples include Wei and Hirzinger [3], Samaras et al [31], Brooks et al [8] and Stewart and Langer [33].

2.1.3 A sequence of 2d images

Given a sequence of images, wherein a rigid object is moving, we can describe the change in image plane projection of its features over time. This will enable us to investigate the way 3d motion affects 2d images.

A simple model of discrete movement includes rotation and translation. At time τ_k, a 3d point $\mathbf{p}$ in the CCS under constant translation would undergo the following transformation:

$$\mathbf{p}(\tau_k) = \mathbf{R}(\tau_k)\mathbf{p}(\tau_{k-1}) + \mathbf{t}(\tau_k), \tag{2.6}$$

where $\mathbf{R}$ is a 3x3 orthonormal matrix representing a rotation about the origin and $\mathbf{t} = [t_x\ t_y\ t_z]^T$ is the vector of 3d translation of the point.

Applying the simple pinhole projection model of (2.3) to (2.6) yields the motion-projection model

$$\mathbf{z}(\tau_k) = \begin{bmatrix} z_x(\tau_k) \\ z_y(\tau_k) \end{bmatrix} \tag{2.7}$$

$$= \frac{f}{f + t_z(\tau_k) + \mathbf{r}_z^T(\tau_k)\mathbf{p}(\tau_{k+1})} \begin{bmatrix} t_x(\tau_k) + \mathbf{r}_x^T(\tau_k)\mathbf{p}(\tau_{k+1}) \\ t_y(\tau_k) + \mathbf{r}_y^T(\tau_k)\mathbf{p}(\tau_{k+1}) \end{bmatrix}. \tag{2.8}$$

This equation allows us to study the effect that rotation and translation has on the projected points of a rigid body:

Imagine a point $\mathbf{p}_a(\tau_{k-1}) = (0, 0, d_z)$. From (2.3), this point projects to $\mathbf{z}_a(\tau_{k-1}) = (0, 0)$. We now translate this point parallel with the x-axis at time τ_k: $\mathbf{t} = (t_x, 0, 0)$. From (2.8), only $z_x(\tau_k)$ will be affected: the point will appear to move horizontally.

Now image the same point is instead rotated by a small angle around the vertical axis. Since a rotation around the y-axis can be described using the rotation matrix

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_x^T \\ \mathbf{r}_y^T \\ \mathbf{r}_z^T \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}, \tag{2.9}$$

it is clear that only $\mathbf{r}_x$ and $\mathbf{r}_z$ will have an effect on the new position of the point, and according to (2.8), the only effect on the observation would be a slight horizontal movement, since $p_y(\tau_{k-1}) = 0$. Therefore, we have a duality, where two different states of motion produce two indistinguishable observations.

However if we take the motion of another point's projection into account, the picture changes: For example, take the point $\mathbf{p}_b(\tau_{k-1}) = (0, d_y, 0)$. If this point lies on the same rigid object as $\mathbf{p}_a(\tau_{k-1})$, it must undergo the same translation and rotation. From (2.3), this point projects to $\mathbf{z}_b(\tau_{k-1}) = (0, \delta_y)$. When the horizontal translation is applied to $\mathbf{p}_b$, (2.8) shows that $z_x(\tau_k)$ changes, so that a noticeable horizontal movement is again observed. However, when the rotation around the vertical y-axis is applied instead, (2.8) shows that the point's 3d position remains the same, and consequently no effect is noticeable on the observation.

This effect can help us discriminate between two types of motions. If we are able to take more observations into account during our estimation, we are able to more discriminate with greater success between different 3d motions. Figure 2.2 illustrates this concept: The top-left graph shows the 3d behaviour of two points while undergoing a simple rotation. The bottom-left graph shows the behaviour of their 2d projections during the rotation. The top-right graph shows the 3d behaviour these same two points during a simple translation, and the bottom-right graphs shows their 2d projection's behaviour. It is clear from this figure how the same point might behave similarly under different types

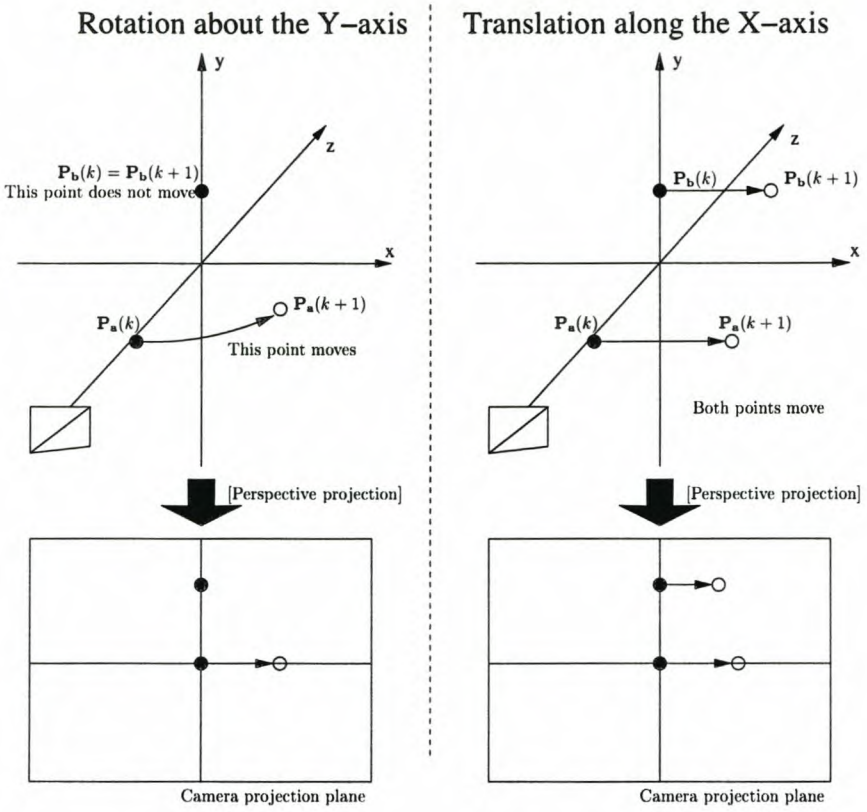


Figure 2.2: *The similarity between the projection of different motions.*

of motion, and also how the incorporation of more points can be used to get a better idea of the type of motion.

The following sections discuss some of the approaches taken in the SfM field in the past. Note that each one considers more than one point at a time, and incorporates more than one frame in the determination of structure from motion.

2.2 An overview of current SfM research

Current methods used in SfM include recovery of the fundamental matrix (from stereovision research), the use of orthographic camera models, geometric algebra models [22], statistical methods [14] and the Kalman filtering approach. Three of these methods are discussed below.

2.2.1 Stereovision

Stereovision, which is usually considered a sister field to SfM, revolves around the disparity map. Given the two images produced by the two cameras, a disparity map relates any feature in one image to its match in the other image. If the camera pair is calibrated so

that the *intrinsic*[1] and *extrinsic*[2] parameters are known, the disparity map can be used to reconstruct the scene under observation as an absolute-scale 3d depth map.

Of interest to Structure from Motion research is the matrix which relates the orientation between two cameras. If this matrix can be determined for two cameras from the 2d feature matches, without any required knowledge of the 3d structure of the scene, it can be directly applied to the SfM problem. It turns out that this matrix can be computed. It is discussed below:

Given the intrinsic camera parameters, a matrix called the essential matrix $[\mathbf{E}]$ can be calculated. It is related to the motion parameters $\{[\mathbf{R}], \mathbf{t}\}$ (where $\mathbf{t} \neq \mathbf{0}$) by:

$$[\mathbf{E}] = [\mathbf{R}][\mathbf{t}_\times], \tag{2.10}$$

where $[\mathbf{R}]$ describes the rotation between the two cameras and $\mathbf{t}_\times$ is an anti-symmetric matrix composed of the components of the translation vector $\mathbf{t}$:

$$\mathbf{t}_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}. \tag{2.11}$$

The essential matrix is related to the disparity map through the relation

$$[\mathbf{x}'_n]^T[\mathbf{E}][\mathbf{x}_n] = 0, \tag{2.12}$$

where $\mathbf{x}_n$ is the nth homogeneous image point for the first image, and $\mathbf{x}'_n$ is its matching homogeneous image point in the other image.

If the intrinsic parameters of the cameras are unknown, the generalised *fundamental* matrix can be calculated, which is related to the essential matrix by

$$[\mathbf{F}] = [\mathbf{C}^{-1}]^T[\mathbf{E}][\mathbf{C}^{-1}], \tag{2.13}$$

where $\mathbf{C}$ is the intrinsic parameter matrix.

It holds the same relation to the the matched homogeneous points as (2.12):

$$[\mathbf{x}'_n]^T[\mathbf{F}][\mathbf{x}_n] = 0. \tag{2.14}$$

Both of these matrices have 9 elements, but is constrained to 7 degrees of freedom being defined up to an arbitrary scale factor and having rank 2. Given

$$\mathbf{F} = \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix}, \tag{2.15}$$

---

[1]The intrinsic camera parameters include the focal length and lens distortion, as well as CCD pixel size information.

[2]The extrinsic parameters describe the location and orientation of the camera within a reference coordinate system.

we can rearrange (2.14) as

$$\mathbf{Mf} = 0 \tag{2.16}$$

$$\begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ & & & & \vdots & & & & \\ x'_Nx_N & x'_Ny_N & x'_N & y'_Nx_N & y'_Ny_N & y'_N & x_N & y_N & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_9 \end{bmatrix} = \mathbf{0}. \tag{2.17}$$

This enables us to calculate $\mathbf{f}$ in a least squares minimisation as

$$\min_{\mathbf{f}} \|\mathbf{Mf}\|^2, \tag{2.18}$$

subject to $\|\mathbf{f}\| = 1$. A solution to (2.18) is the eigenvector of $\mathbf{M}^T\mathbf{M}$ with the minimum eigenvalue. Methods for solving this equation has been developed by Tsai [36], Zhang [42], Trucco & Verri [35] as well as Luong et al [24].

The discussion above shows that we are able to extract the motion $\{[\mathbf{R}], \mathbf{t}\}$ of the properly calibrated camera from one frame to another, using only the feature matches between images. Additionally, if we regard the images in a video sequence as a "necklace" of cameras stringed together by small rotations and translations, we can calculate the essential matrix between each two consecutive cameras to produce a history of the rotation and scaled translation of the camera in relation to the static scene. Once we have extracted the motion, we can use this information to find the 3d structure represented by the points.

However, the calculation of the essential matrix, and in the uncalibrated case the fundamental matrix, can be very sensitive to noise, especially when the motion between subsequent frames are small [16], as in the case of Structure from Motion. Also, when the camera only rotates between frames, the above solution is not valid, and other methods must be used to extract the motion information. Luong et al [24] and Luong and Faugeras [23] investigate different methods as well as the stability of the fundamental matrix. This approach does however remain a promising area of SfM research.

2.2.2 Orthogonal projection and the SVD

One way of simplifying the Structure from Motion task is to assume a simpler camera model. One such a model is the orthographic projection model, which was defined in (2.4). Since this camera model is linear, it allows for a linear solution.

In [34] Tomasi and Kanade introduce a factorisation batch algorithm to process a monocular image sequence of a rigid object. Measurements, consisting of the tracked

features on a rigid object, are concatenated together in a single matrix

$$\mathbf{W} = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1N} \\ \vdots & \vdots & & \vdots \\ u_{F1} & u_{F1} & \dots & u_{FN} \\ v_{11} & v_{12} & \dots & v_{1N} \\ \vdots & \vdots & & \vdots \\ v_{F1} & v_{F2} & \dots & v_{FN} \end{bmatrix}, \tag{2.19}$$

where (u_{fn}, v_{fn}) is the nth point's 2d orthographic projection in frame f, out of N points and F frames.

The 3d location of the nth point in the fth frame is related to its original 3d location in the first frame by the rotation

$$\mathbf{R}_f = \begin{bmatrix} r_{11,f} & r_{12,f} & r_{13,f} \\ r_{21,f} & r_{22,f} & r_{23,f} \\ r_{31,f} & r_{32,f} & r_{33,f} \end{bmatrix}, \tag{2.20}$$

and translation $\mathbf{t}_f = (t_{xf}, t_{yf}, t_{zf})$, which together aligns the object's coordinate system in frame f with its initial coordinate system.

Therefore, $\mathbf{W}$ can be written in terms of a motion matrix $\mathbf{M}$ and a shape matrix $\mathbf{S}$

$$\mathbf{W} = \mathbf{MS}, \tag{2.21}$$

where

$$\mathbf{M} = \begin{bmatrix} r_{11,1} & r_{12,1} & r_{13,1} & t_{x,1} \\ & \vdots & & \vdots \\ r_{11,F} & r_{12,F} & r_{13,F} & t_{x,F} \\ r_{21,1} & r_{22,1} & r_{23,1} & t_{y,1} \\ & \vdots & & \vdots \\ r_{21,F} & r_{22,F} & r_{23,F} & t_{y,F} \end{bmatrix}, \tag{2.22}$$

and

$$\mathbf{S} = \begin{bmatrix} s_{x1} & & s_{xN} \\ s_{y1} & \dots & s_{yN} \\ s_{z1} & & s_{zN} \\ 1 & & 1 \end{bmatrix}. \tag{2.23}$$

Looking at $\mathbf{M}$, it is clear that this matrix folds the 3d $\mathbf{S}$ matrix into the 2d $\mathbf{W}$ matrix and therefore describes the orthographic projection camera model. It also contains the rotation and translation parameters of the object. However, note that the translation

paramter t_{zf} and the rotation row-vector $[r_{31,f}\ r_{32,f}\ r_{33,f}]$ are missing. The rotation vector is however easy to recover, since the rotation matrix only has three degrees of freedom. The translation t_{zf} is lost.

In order to extract $\mathbf{M}$ and $\mathbf{S}$ from $\mathbf{W}$, the SVD of $\mathbf{W}$ approximated it as

$$\mathbf{W} = \mathbf{U}\boldsymbol{\varepsilon}\mathbf{V}^T. \tag{2.24}$$

A matrix $\mathbf{A}$ is now computed, so that

$$\mathbf{S} = \mathbf{A}^{-1}\boldsymbol{\varepsilon}^{1/2}\mathbf{V}^T \text{ and } \mathbf{M} = \mathbf{U}\boldsymbol{\varepsilon}^{1/2}\mathbf{A}, \tag{2.25}$$

by imposing rotational and translational constraints implied by the nature of $\mathbf{M}$ as well as choosing $\mathbf{R}_1 = \mathbf{I}$. (A more detailed discussion of these steps are given by Costeira in [10])

Therefore a simple post-processing linear operation on batch data recovers the motion and structure of a moving object in relation to a static camera. If we take a close look at the $\mathbf{S}$ and $\mathbf{M}$ matrices, we see that the object's shape and rotation is fully recovered in 3d. However the translation of the object can only be recovered in a direction orthogonal to the optical axis, since any motion parallel to the optical axis produces no change in the projection, and therefore conveys no information. This follows directly from the definition of orthogonal projection, where the z-axis information is discarded. Another subtle point is that no reconstruction of shape is possible if the object only translates, but does not rotate.

Morita and Kanade extends this algorithm in [26] by adopting a sequential factorisation method. This modifies the original algorithm from a batch solution to a recursive solution, suitable for real-time implementation. It also lowers the computational cost of the singular value decomposition and speeds up the algorithm.

Poelman and Kanade [30] modifies the original algorithm by assuming a para-perspective projection model. Para-perspective projection more closely approximates the perspective camera model, modelling several effects neglected in orthographic projection. A big advantage is that it retains the linear algebraic properties.

2.2.3 Perspective projection and the Kalman filter

As discussed in the previous section, some earlier work simplified the SfM problem by assuming an orthographic projection camera model. This converts the projection function from a non-invertible *non-linear* function to a non-invertible *linear* function, which simplifies the problem somewhat, but does not allow the reconstruction of object motion parallel to the optical axis of the camera. All the depth information inherent in pure translation is lost using this method. A perspective projection camera model may be

used instead. However this projection is non-invertible and non-linear, which precludes the use of linear techniques in formulating a solution to the problem.

In [7], Broida et al introduces an algorithm based on this camera model. It uses a Kalman filter to process the image plane data recursively in order to calculate the structure and motion of the object. The Kalman filter can be used as a hidden-state estimator, which recursively compares measurements based on its estimate of the current hidden state to actual measurements in order to update the estimate. In the SfM case, the hidden state corresponds to the unknown structure and motion, while the measurements correspond to the 2d image plane measurements. However, the Kalman filter is a linear filter, and the Extended Kalman filter was used by Broida et al instead. The EKF is an approximation of the Kalman filter for non-linear problems. It is discussed in some detail in chapter 3.

The rigid object's centre is expressed as

$$\mathbf{s}_R(\tau) = \begin{bmatrix} x_R(\tau) & y_R(\tau) & z_R(\tau) \end{bmatrix}^T, \tag{2.26}$$

where the origin of the coordinate system coincides with the focal point of the camera. In terms of this, the object is assigned five variables representing its translation vector and translational velocity vector,

$$\mathbf{t} = \begin{bmatrix} x_R(\tau)/z_R(\tau) \\ y_R(\tau)/z_R(\tau) \\ \dot{x}(\tau)/z_R(\tau) \\ \dot{y}(\tau)/z_R(\tau) \\ \dot{z}(\tau)/z_R(\tau) \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{bmatrix}, \tag{2.27}$$

where (s_1, s_2) represents the image plane projected translation of the object, in terms of its 3d translation, and the vector (s_3, s_4, s_5) represents the normalised translational velocity.

Four state variables describe the rotation of the object using a quaternion[3] representation, and three variables represent the object's rotational velocity of the object around its principle axii:

$$\mathbf{r} = \begin{bmatrix} q_1(\tau) \\ q_2(\tau) \\ q_3(\tau) \\ q_4(\tau) \\ \omega_x(\tau) \\ \omega_y(\tau) \\ \omega_z(\tau) \end{bmatrix} = \begin{bmatrix} s_6 \\ s_7 \\ s_8 \\ s_9 \\ s_{10} \\ s_{11} \\ s_{12} \end{bmatrix}, \tag{2.28}$$

[3]The quaternion and its relation to an orthogonal rotation matrix is given in section 4.3.1. Faugeras [11] contains more detailed information on the quaternion and its use in computer vision.

where the quaternion is calculated as $\mathbf{q}(\tau) = s_6\mathbf{i} + s_7\mathbf{j} + s_8\mathbf{k} + s_9$, and the rotational velocity vector as $\boldsymbol{\omega} = (s_{10}, s_{11}, s_{12})$.

Each object point n in the set of N is assigned three state variables, representing its 3d location:

$$\mathbf{p}_n = \begin{bmatrix} x_n/z_R(\tau) \\ y_n/z_R(\tau) \\ z_n/z_R(\tau) \end{bmatrix} = \begin{bmatrix} s_{3n+10} \\ s_{3n+11} \\ s_{3n+12} \end{bmatrix}, \tag{2.29}$$

which represents the normalised 3d location of the point at time τ.

The EKF requires the time-derivative of the state variable vector, which is calculated for the system

$$\mathbf{x} = [\mathbf{t},\ \mathbf{r},\ \mathbf{p}_1, \ldots,\ \mathbf{p}_N], \tag{2.30}$$

and defined in terms of the state variables themselves:

$$\dot{\mathbf{s}}(\tau) = \begin{bmatrix} \dot{x}/z_R(\tau) - [x_R(\tau)/z_R(\tau)][\dot{z}/z_R(\tau)] \\ \dot{y}/z_R(\tau) - [y_R(\tau)/z_R(\tau)][\dot{z}/z_R(\tau)] \\ -[\dot{x}/z_R(\tau)][\dot{z}/z_R(\tau)] \\ -[\dot{y}/z_R(\tau)][\dot{z}/z_R(\tau)] \\ -[\dot{z}/z_R(\tau)][\dot{z}/z_R(\tau)] \\ 0.5(\omega_z q_2 - \omega_y q_3 + \omega_x q_4) \\ 0.5(-\omega_z q_1 + \omega_x q_3 + \omega_y q_4) \\ 0.5(\omega_y q_1 - \omega_x q_2 + \omega_z q_4) \\ 0.5(-\omega_x q_1 - \omega_y q_2 - \omega_z q_3) \\ 0 \\ 0 \\ 0 \\ \vdots \\ -[x_n/z_R(\tau)][\dot{z}/z_R(\tau)] \\ -[y_n/z_R(\tau)][\dot{z}/z_R(\tau)] \\ -[z_n/z_R(\tau)][\dot{z}/z_R(\tau)] \\ \vdots \end{bmatrix} = \begin{bmatrix} s_3 - s_1 s_5 \\ s_4 - s_2 s_5 \\ -s_3 s_5 \\ -s_4 s_5 \\ -s_5^2 \\ 0.5(s_{12}s_7 - s_{11}s_8 + s_{10}s_9) \\ 0.5(-s_{12}s_6 - s_{10}s_8 + s_{11}s_9) \\ 0.5(s_{11}s_6 - s_{10}s_7 + s_{12}s_9) \\ 0.5(-s_{10}s_6 - s_{11}s_7 + s_{12}s_8) \\ 0 \\ 0 \\ 0 \\ \vdots \\ -s_{3n+10}s_5 \\ -s_{3n+11}s_5 \\ -s_{3n+12}s_5 \\ \vdots \end{bmatrix} \tag{2.31}$$

The EKF also requires a measurement function, which is again derived each point and written in terms of the state variables:

$$\mathbf{z}_n = \frac{1}{1 + \mathbf{r}_z^T \mathbf{p}_n} \begin{bmatrix} s_1 + \mathbf{r}_x^T \mathbf{p}_n \\ s_2 + \mathbf{r}_y^T \mathbf{p}_n \end{bmatrix}, \tag{2.32}$$

where $\mathbf{R} = [\mathbf{r}_x\ \mathbf{r}_y\ \mathbf{r}_z]^T$ is the rotation matrix calculated from the quaternion during each frame.

The main advantage of this system lies in the use of the more realistic camera model which allows structure and motion calculation even from pure 3d translation. Recall that this was not possible with an orthographic camera. Another advantage lies in the recursive nature of the Kalman filter, which makes the algorithm more suitable for real-time applications than a batch algorithm. The Kalman filter also allows the algorithm to succeed under high process- as well as measurement noise levels.

The main disadvantage however lies in the use of the EKF. The EKF is a sub-optimal implementation of the Kalman filter. It requires very good initial information, typically acquired by running an Iterated Extended Kalman filter on a batch consisting of the first couple of frames in the sequence (as was implemented by Broida et al). Secondly, while the object's motion is allowed to change over time, sudden large changes causes the EKF to fail, and requires a subsequent re-initialisation before normal operation can proceed.

2.3 The proposed algorithm

In this thesis, the perspective camera model is used, along with a non-linear Kalman filter approximation, using a similar approach as described in the previous section.

The proposed algorithm splits the motion and structure estimation into two separate state variables, which is implemented as a dual Unscented Kalman filter instead of the usual EKF. The Unscented Kalman filter was introduced by Julier et al [20], and produces more accurate results than the EKF. The dual-UKF, which is an application of the UKF to specific problems, was introduced by Wan and Van der Merwe [37].

The state variable choice in the proposed algorithm follows the same reasoning as the original filter by Broida et al, but implements the structure model introduced by Jebara et al [16]. This structure model enforce the rigidity constraint by reducing the number of state variables per point to one.

Chapter 3 discussed the Kalman filter and its non-linear approximations, including the Unscented Kalman filter. Chapter 4 discusses the internals of the proposed algorithm, followed by chapter 5 which discusses the experimental results.

Chapter 3

The Kalman filter

The Kalman filter is, to quote Chui [9], "a linear, unbiased, and minimum error variance recursive algorithm that estimates, optimally, the unknown state of a dynamical system from noisy data taken at discrete time intervals". This makes it ideal for application to the SfM problem. The main complication is that the SfM problem is nonlinear under the perspective camera model.

Fortunately, various nonlinear approximations of the linear Kalman filter are available. Two of these, namely the Extended Kalman filter and the Unscented Kalman filter, are discussed in this chapter. The EKF is discussed because it is a standard way of solving the SfM problem, as used in [16], [41], [43], [7] and [5].

This thesis proposes the use of the Unscented Kalman filter in Structure from Motion instead, since it is a better approximation of the Kalman filter, and has proven itself in a number of situations. Therefore the Kalman filter is discussed next in further detail, followed by the EKF and concluding with the UKF.

3.1 The Kalman filter

The Kalman filter's role in the SfM algorithms mentioned above is to function as a state estimator. Using measurements produced by an unkown state, the goal is to produce an estimate of the unkown state. In our case, the unknown state consists of a rigid object's structure and motion information. The measurements available to us is the tracked 2d image plane coordinates that the object produces over time.

Consider an unknown discrete linear system

$$\mathbf{x}(k+1) = \boldsymbol{\mathcal{F}}(k)\mathbf{x}(k) + \mathbf{q}(k) \tag{3.1}$$

$$\mathbf{y}(k) = \boldsymbol{\mathcal{H}}(k)\mathbf{x}(k) + \mathbf{r}(k), \tag{3.2}$$

where $\mathbf{x}$ is the unobservable state of the system and $\mathbf{y}$ is the observable measurements. $\boldsymbol{\mathcal{F}}(k)$ is the time propagation transform, $\boldsymbol{\mathcal{H}}(k)$ is the measurement transform, $\mathbf{q}(k)$ represents process noise and $\mathbf{r}(k)$ represents measurement noise. Equation (3.1) is the process

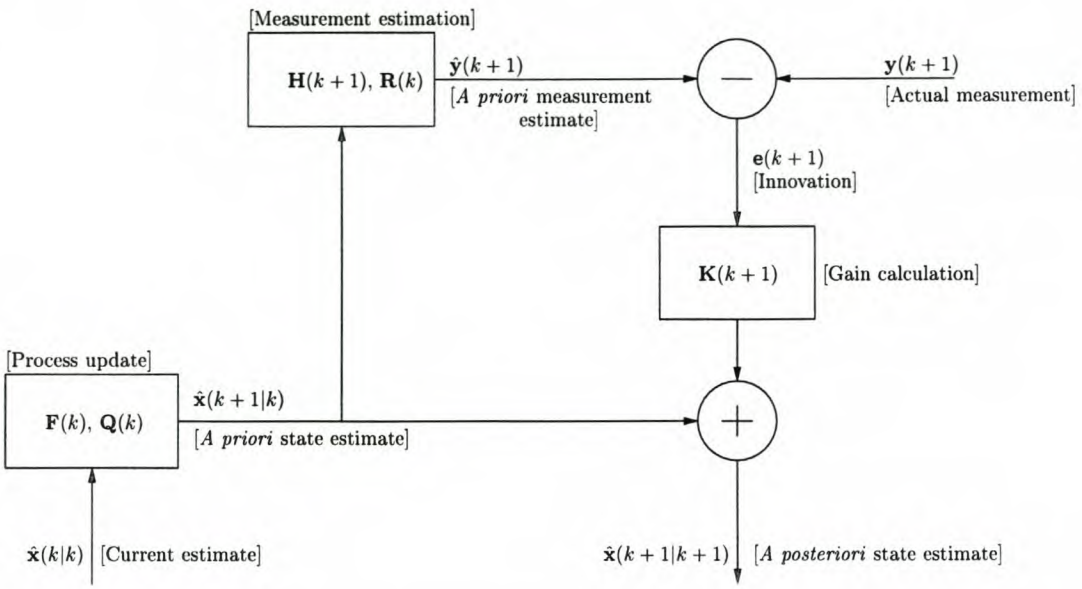


Figure 3.1: *A simple state estimation loop.*

update equation, and (3.2) is the measurement equation. In this system, we have complete knowledge of $\mathbf{y}$. We also have a model $\mathbf{F}$ of $\mathcal{F}$, a model $\mathbf{H}$ of $\mathcal{H}$, and assume some knowledge of the properties of $\mathbf{q}$ and $\mathbf{r}$.

An overview of the flow of the Kalman filter is shown figure 3.1: The state estimate at time k is updated using the process update model, and an estimate of the measurement is then formed via the measurement estimate model. The difference between the estimated and actual measurements, called the *innovation*, is multiplied by a *Kalman gain*, and is used to improve the *a priori* state estimate to produce the final *a posteriori* estimate at time $k + 1$.

The Kalman gain is calculated in the Kalman filter by combining an estimate $\hat{\mathbf{P}}(k)$ of the state error covariance with the estimated measurement error covariance. The state error covariance is given by the expected value of the state error,

$$\mathbf{P}(k) = E\{[\hat{\mathbf{x}}(k) - \mathbf{x}(k)][\hat{\mathbf{x}}(k) - \mathbf{x}(k)]^T\}. \tag{3.3}$$

where $\hat{\mathbf{x}}(k)$ is the *a posteriori* estimate of the state $\mathbf{x}(k)$.

3.1.1 The Kalman filter loop

The Kalman filter is an optimal linear filter that recursively updates the current optimal estimate, given a new observation. Prior knowledge of the measurement and process models as well as the measurement and process noise is assumed.

Each loop or iteration of the Kalman filter consists of a process propagation update and a measurement estimate. The process update step merely propagates the state estimate

and the error covariance matrix in time using the known system dynamics:

$$\hat{\mathbf{x}}(k+1|k) = \mathbf{F}(k)\hat{\mathbf{x}}(k|k) \tag{3.4}$$

$$\hat{\mathbf{P}}(k+1|k) = \mathbf{F}(k)\hat{\mathbf{P}}(k|k)\mathbf{F}^T(k) + \mathbf{Q}(k), \tag{3.5}$$

where (3.5) is the *Ricatti* equation which propagates the estimated error covariance matrix. $\mathbf{Q}(k)$ is the process error covariance matrix. The result of the update step is the *a priori* versions of the estimates state and error covariance matrices.

The measurement estimate step uses the *a priori* state estimate to calculate the measurement estimate:

$$\hat{\mathbf{y}}(k+1|k) = \mathbf{H}(k+1)\hat{\mathbf{x}}(k+1|k). \tag{3.6}$$

The difference between the estimated and observed measurements is called the *innovation*:

$$\mathbf{e}(k+1) = \mathbf{y}(k+1) - \hat{\mathbf{y}}(k+1|k) \tag{3.7}$$

$$= \mathbf{y}(k+1) - \mathbf{H}(k+1)\hat{\mathbf{x}}(k+1|k). \tag{3.8}$$

The final step computes the estimated state improvement

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{K}(k+1)\mathbf{e}(k+1), \tag{3.9}$$

and the improved estimated error covariance

$$\hat{\mathbf{P}}(k+1|k+1) = \mathbf{K}(k+1)\mathbf{R}(k+1)\mathbf{K}^T(k+1)+ [\mathbf{I} - \mathbf{K}(k+1)\mathbf{H}(k+1)]\hat{\mathbf{P}}(k+1|k)[I - \mathbf{K}(k+1)\mathbf{H}(k+1)]^T, \tag{3.10}$$

where $\mathbf{R}(k)$ is the measurement error covariance matrix, and $\mathbf{K}(k+1)$ is the *Kalman gain*, which in effect controls how much the innovation value contributes to the state estimate. $\mathbf{K}(k+1)$ is calculated using:

$$\mathbf{K}(k+1) = \hat{\mathbf{P}}(k+1|k)\mathbf{H}^T(k+1)\left[\mathbf{H}(k+1)\hat{\mathbf{P}}(k+1|k)\mathbf{H}^T(k+1) + \mathbf{R}(k+1)\right]^{-1}. \tag{3.11}$$

Both the measurement and process noise is assumed to be zero-mean and white. $\mathbf{Q}(k)$ and $\mathbf{R}(k)$ are related to $\mathbf{q}(k)$ and $\mathbf{r}(k)$ respectively by

$$E[\mathbf{q}(k)\mathbf{q}^H(l)] = \begin{cases} \mathbf{Q}(k), & k = l \\ \mathbf{0}, & k \neq l \end{cases} \tag{3.12}$$

$$E[\mathbf{r}(k)\mathbf{r}^H(l)] = \begin{cases} \mathbf{R}(k), & k = l \\ \mathbf{0}, & k \neq l \end{cases}. \tag{3.13}$$

Intuitively, the influence of the two types of noise can be seen from the equations [39]: From (3.5), it is seen that the process noise is added to the estimated error covariance

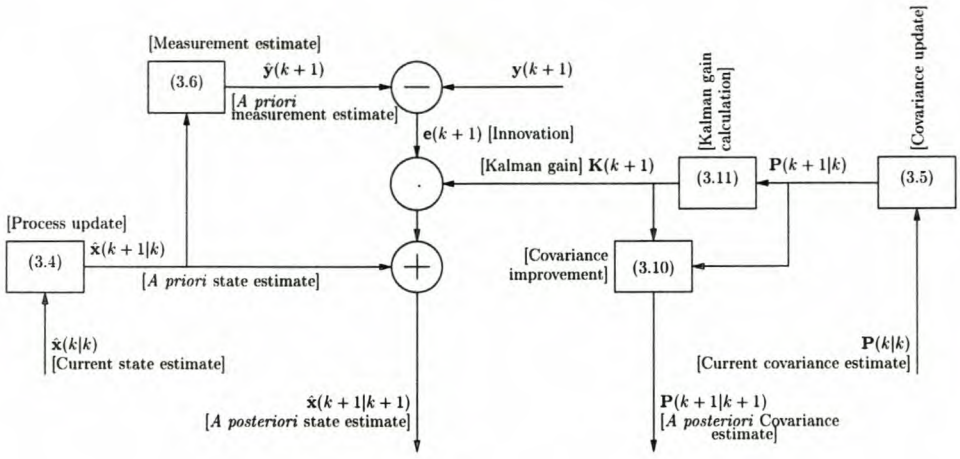


Figure 3.2: *The Kalman filter's recursive loop.*

matrix. This, in turn, increases the Kalman gain (3.11), which causes the innovation to have a greater influence on the final estimate (3.9). A large process noise therefore minimises the effect of the current estimate, as if the Kalman filter distrusts the estimate.

From (3.11) it is seen that the Kalman gain is inversely proportional to the measurement noise. A large measurement noise value therefore decreases the innovation's effect on the state estimate, as if the Kalman filter distrusts the measurements.

For example, if the measurement noise $\mathbf{R}$ is zero, the Kalman gain will equal the inverse of $\mathbf{H}(k+1)$. This will in turn result in (3.9) treating the innovation as being optimally correct.

The correct balance of process noise and covariance noise causes the filter to correctly incorporate the observations into the estimate to optimally converge in the presence of noise. The process of choosing the noise covariances is called tuning.

A flow diagram of the Kalman filter can be seen in figure 3.2.

3.2 The Extended Kalman filter

As discussed in the previous section, the Kalman filter optimally estimates the state of an unknown *linear* dynamic system. The Kalman filter estimate is a multi-dimensional Gaussian distribution with the mean given by the estimated state, and the variance given by the state error covariance matrix. This distribution is required to undergo the same process and measurement transformations as a whole. If these transformations are non-linear, the resultant distribution is not Gaussian anymore, invalidating the Kalman filter's estimate. Therefore, the Kalman filter is limited to linear problems.

Consider the following non-linear *continuous* time dynamic system:

$$\frac{\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{q}(t) \tag{3.14}$$

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t), t) + \mathbf{r}(t). \tag{3.15}$$

It is clear from the previous section that the Kalman filter cannot be directly applied to this situation. Many real-world problems are however nonlinear. Various attempts have therefore been made to modify the original Kalman filter to allow it to be applied to non-linear problems.

Today, the most widely used implementation is the Extended Kalman filter. The EKF modifies the original Kalman filter by linearising the system dynamics around the current estimate.

3.2.1 The EKF modifications

The EKF linearises the process update and measurement estimate equations around the current state estimate. This in effect replaces the forward propagation equation (3.4) with

$$\hat{\mathbf{x}}(k+1|k) = \hat{\mathbf{x}}(k|k) + \int_{t_k}^{t_{k+1}} \mathbf{f}(\hat{\mathbf{x}}(k|k), \tau) d\tau, \tag{3.16}$$

and the measurement estimate equation (3.6) with

$$\hat{\mathbf{y}}(k+1|k) = \mathbf{h}(\hat{\mathbf{x}}(k+1|k), t_{k+1}). \tag{3.17}$$

For the propagation update and measurement estimate of the error covariance matrix, the Jacobian of $\mathbf{f}(t)$ and $\mathbf{h}(t)$ will replace $\mathbf{F}(k)$ and $\mathbf{H}(k)$ respectively in equations (3.5), (3.10) and (3.11),

$$\mathbf{F}(k) := \frac{d\mathbf{f}(\mathbf{x}, t_k)}{d\mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}(k|k)} \tag{3.18}$$

$$\mathbf{H}(k) := \frac{d\mathbf{h}(\mathbf{x}, t_{k+1})}{d\mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}(k+1|k)}. \tag{3.19}$$

A flow diagram of the Extended Kalman filter can be seen in figure 3.3. This figure clearly illustrates the shortcomings of the EKF: One set of process update and measurement estimate equations is used to propagate the estimated state and calculate the estimated measurement respectively. A *different* set is used in the state error and measurement error covariance calculation. If the linearisation takes place in an area where the state becomes highly non-linear, or for instance near a discontinuity, the covariances do not propagate in the same manner as the means.

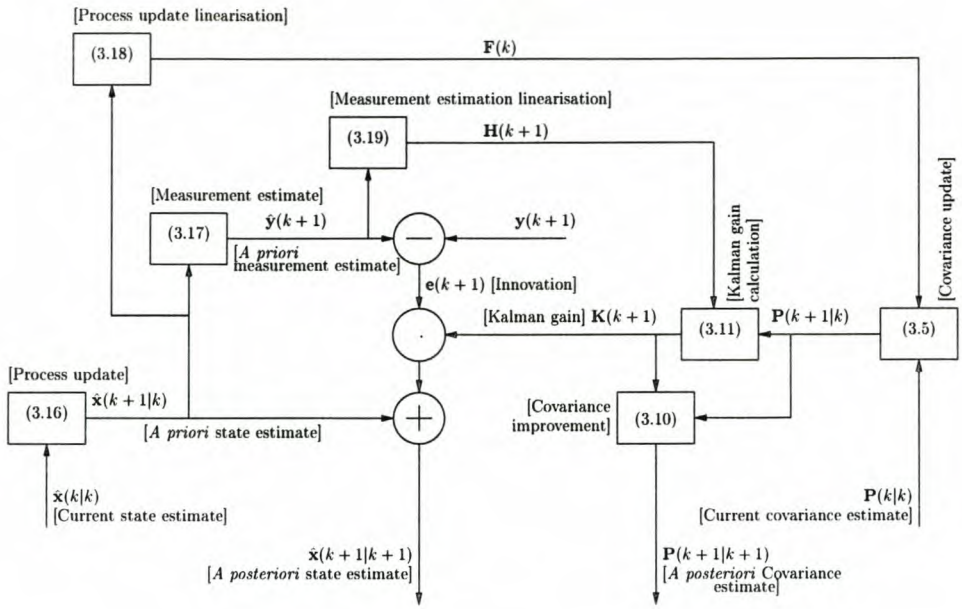


Figure 3.3: *The Extended Kalman filter's recursive loop.*

3.3 The Unscented Kalman filter

The Unscented Kalman filter was proposed by Julier and Uhlman [19]. It employs the *Unscented Transform* to accurately compute the new mean and covariance of the state estimate distribution undergoing a non-linear transform, up to the 3rd order Taylor series expansion [37]. This promises greater accuracy and stability in real-world non-linear applications, such as the SfM problem. The Unscented Transform, on which the UKF is based, is discussed first, followed by the UKF.

3.3.1 The Unscented Transform

Consider a random variable undergoing a nonlinear transformation:

$$\mathbf{y} = H(\mathbf{x}), \tag{3.20}$$

where $\mathbf{x}$ is the original random variable, $H(\cdot)$ is the non-linear transformation, and $\mathbf{y}$ is the transformed random variable.

The Unscented Transform specifies the random variable in terms of a minimal set of sample points, which are propagated through the non-linear transform and recombined to produce an approximation of the transformed random variable.

For non-Gaussian variables, this approach is in general second order accurate [37]. It is third order accurate for Gaussian variables, and achieves this with a computational complexity similar to that of the EKF, which is only first order accurate.

The following subsections discuss the working of the UT. The first subsection shows how the sigma-points are calculated for a given random variable. The second shows how the sigma points are propagated through the non-linear function. The last shows the transformed sigma points are recombined to form a new random variable.

Calculating sigma points from a random variable

The mean $\bar{\mathbf{x}}$ and covariance $\mathbf{P_x}$ of the Gaussian random variable $\mathbf{x}$, with dimension L, is used to generate the following matrix of sigma vectors $\boldsymbol{\chi}$:

$$\boldsymbol{\chi}_0 = \bar{\mathbf{x}} \tag{3.21}$$

$$\boldsymbol{\chi}_i = \bar{\mathbf{x}} + \boldsymbol{\Phi}_i \qquad i = 1, \ldots, L \tag{3.22}$$

$$\boldsymbol{\chi}_i = \bar{\mathbf{x}} - \boldsymbol{\Phi}_{i-L} \qquad i = L+1, \ldots, 2L, \tag{3.23}$$

where

$$\frac{1}{L}\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \mathbf{P_x}. \tag{3.24}$$

$\boldsymbol{\Phi}_i$ is the i'th column of $\boldsymbol{\Phi}$ in the text above.

Along with the sigma vectors, a set of weights w is generated:

$$w_0^m = 0 \tag{3.25}$$

$$w_0^c = 2 \tag{3.26}$$

$$w_i^m = w_i^c = \frac{1}{2L} \qquad i = 1, \ldots, 2L, \tag{3.27}$$

where the superscript m indicates a weight associated with the mean and the superscript c indicates a weight associated with covariance.

It is useful to indicate this mapping of a random variable to its sigma point set as

$$\boldsymbol{\chi} = \Upsilon(\bar{\mathbf{x}}, \mathbf{P_x}). \tag{3.28}$$

Non-linear sigma-point propagation

The aim of the Unscented Transform is to find an approximation of the resultant random variable after it has undergone a possibly nonlinear transformation. The solution in the sigma vector "space" is to transform each sigma vector with the nonlinear transform to produce a sigma vector set $\boldsymbol{\gamma}$:

$$\boldsymbol{\gamma}_i = H(\boldsymbol{\chi}_i) \qquad i = 0, \ldots, 2L\ , \tag{3.29}$$

where $H(\cdot)$ is the non-linear function.

Calculating a random variable from sigma-points

We now have the transformed sigma point set. A weighted sum of this set is used to calculate the transformed Gaussian variable $\mathbf{y}$ with a mean $\bar{\mathbf{y}}$,

$$\bar{\mathbf{y}} \approx \sum_{i=0}^{2L} w_i^m \boldsymbol{\gamma}_i \tag{3.30}$$

$$= \frac{1}{2L} \sum_{i=1}^{2L} \boldsymbol{\gamma}_i \tag{3.31}$$

and covariance $\mathbf{P}_\mathbf{y}$,

$$\mathbf{P}_\mathbf{y} \approx \sum_{i=0}^{2L} w_i^c \left[\boldsymbol{\gamma}_i - \bar{\mathbf{y}}\right] \left[\boldsymbol{\gamma}_i - \bar{\mathbf{y}}\right]^T \tag{3.32}$$

$$= 2[\boldsymbol{\gamma}_0 - \bar{\mathbf{y}}][\boldsymbol{\gamma}_0 - \bar{\mathbf{y}}]^T + \frac{1}{2L} \sum_{i=1}^{2L} \left[\boldsymbol{\gamma}_i - \bar{\mathbf{y}}\right] \left[\boldsymbol{\gamma}_i - \bar{\mathbf{y}}\right]^T . \tag{3.33}$$

These operations will be referred to using the notation $\Upsilon^+(\cdot)$ in the rest of this thesis, so that

$$(\bar{\mathbf{y}},\ \mathbf{P}_\mathbf{y}) = \Upsilon^+(\boldsymbol{\gamma}). \tag{3.34}$$

The Scaled Unscented Transform

The UT discussed here is the basic UT presented by Julier and Uhlmann [18] and Julier et al [20]. More recently, a more complex version of the UT has been developed, which is called the Scaled Unscented transform.

This transform employs extra parameters to enable a scaled set of sample points to match the mean and covariance of a non-Gaussian distribution, while preserving second-order accuracy. More on this can be found in work by Julier [17].

Conclusion

In general, the covariance of a distribution is defined in terms of its variances. If the distribution is transformed (non-linearly or otherwise), it makes sense that the variances are transformed as well. Therefore, it also makes sense to find the new distribution from the transformed variances.

This is in fact exactly what the UT does: the sigma points represent the variances of the distribution along its main axii. The reason why the UT is still only an *approximation*, is because it tries to find the Gaussian distribution which best fits the new distribution.

3.3.2 The implementation of the UKF

The implementation of the UKF is quite straightforward. Basically, the three Gaussian random variables described by

- mean $\hat{\mathbf{x}}(k)$ and covariance $\hat{\mathbf{P}}(k)$,
- (zero mean and) covariance $\mathbf{Q}(k)$ and
- (zero mean and) covariance $\mathbf{R}(k)$

are converted into their corresponding sigma point sets. These sigma points now take the place of the usual quantities in the Kalman process equations, as described in the UT discussion above, where-after the *a posteriori* distributions are calculated from these transformed sigma points.

In more detail: the UKF first concatenates the various quantities $\hat{\mathbf{x}}(k)$, $\hat{\mathbf{P}}(k)$, $\mathbf{Q}(k)$ and $\mathbf{R}(k)$ into a super state variable $\hat{\mathbf{x}}^a$ and super state covariance matrix $\hat{\mathbf{P}}^a$,

$$\hat{\mathbf{x}}^a(k) = [\hat{\mathbf{x}}(k)\ \mathbf{0}\ \mathbf{0}] \tag{3.35}$$

$$\hat{\mathbf{P}}^a(k) = \begin{bmatrix} \mathbf{P}(k) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}(k) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{R}(k) \end{bmatrix}, \tag{3.36}$$

which is transformed using the forward UT[1].

The algebraic process is as follows: For each time step k

1. The sigma points are calculated:

$$\boldsymbol{\chi}^a(k) = \Upsilon\left(\hat{\mathbf{x}}^a(k), \hat{\mathbf{P}}^a(k)\right) \tag{3.37}$$

$$= [(\boldsymbol{\chi}^x)\ \ (\boldsymbol{\chi}^q)\ \ (\boldsymbol{\chi}^r))]^T. \tag{3.38}$$

2. The state estimate and process noise sigma vectors are propagated using the non-linear propagation function $\mathbf{F}(\cdot)$

$$\boldsymbol{\chi}^x(k+1|k) = \mathbf{F}(\boldsymbol{\chi}^x(k), \boldsymbol{\chi}^q(k), k), \tag{3.39}$$

and the *a priori* estimate can be computed using

$$\left(\hat{\mathbf{x}}(k+1|k), \hat{\mathbf{P}}(k+1|k)\right) = \Upsilon^+(\boldsymbol{\chi}^x(k+1|k)). \tag{3.40}$$

---

[1]Note that the super state variable is merely convenient. The three distributions may just as well have been transformed separately. However, the Matlab code uses this approach, and it is explained as such here.

Making use of (3.30) and (3.32), the *a priori* estimated state and error covariance matrix are explicitly given by

$$\hat{\mathbf{x}}(k+1|k) = \sum_{i=0}^{2L} w_i^m \boldsymbol{\chi}_i^x(k+1|k) \tag{3.41}$$

$$\hat{\mathbf{P}}(k+1|k) = \sum_{i=0}^{2L} w_i^c [\boldsymbol{\chi}_i^x(k+1|k) - \hat{\mathbf{x}}(k+1|k)][\boldsymbol{\chi}_i^x(k+1|k) - \hat{\mathbf{x}}(k+1|k)]^T. \tag{3.42}$$

3. The state estimate and measurement noise sigma vectors are processed with the non-linear measurement equation $\mathbf{H}(\cdot)$:

$$\boldsymbol{\gamma}(k+1|k) = \mathbf{H}(\boldsymbol{\chi}^x(k+1|k), \boldsymbol{\chi}^r(k+1), k+1), \tag{3.43}$$

and the *a priori* measurement estimate is calculated using

$$\hat{\mathbf{y}}(k) = \sum_{i=0}^{2L} w_i^m \boldsymbol{\gamma}_i^x(k+1|k). \tag{3.44}$$

4. The measurement error covariance matrix is computed using

$$\hat{\mathbf{P}}_{\tilde{\mathbf{y}}\tilde{\mathbf{y}}}(k+1|k) = \sum_{i=0}^{2L} w_i^c [\boldsymbol{\gamma}_i(k+1|k) - \hat{\mathbf{y}}(k+1|k)][\boldsymbol{\gamma}_i(k+1|k) - \hat{\mathbf{y}}(k+1|k)]^T, \tag{3.45}$$

and the cross state/observation error covariance matrix is computed using

$$\hat{\mathbf{P}}_{\mathbf{xy}}(k+1|k) = \sum_{i=0}^{2L} w_i^c [\boldsymbol{\chi}_i^x(k+1|k) - \hat{\mathbf{x}}(k+1|k)][\boldsymbol{\gamma}_i(k+1|k) - \hat{\mathbf{y}}(k+1|k)]^T. \tag{3.46}$$

These are then combined, along with the previous calculations to compute the Kalman gain [37]

$$\mathbf{K}(k+1) = \hat{\mathbf{P}}_{\mathbf{xy}}(k+1|k)\hat{\mathbf{P}}_{\tilde{\mathbf{y}}\tilde{\mathbf{y}}}^{-1}(k+1|k), \tag{3.47}$$

which is used to update the state estimate and estimated error covariance matrix

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{K}(k+1)\big(\mathbf{y}(k+1) - \hat{\mathbf{y}}(k+1|k)\big) \tag{3.48}$$

$$\hat{\mathbf{P}}(k+1|k+1) = \hat{\mathbf{P}}(k+1|k) - \mathbf{K}(k+1)\hat{\mathbf{P}}_{\tilde{\mathbf{y}}\tilde{\mathbf{y}}}(k+1|k)\mathbf{K}^T(k+1). \tag{3.49}$$

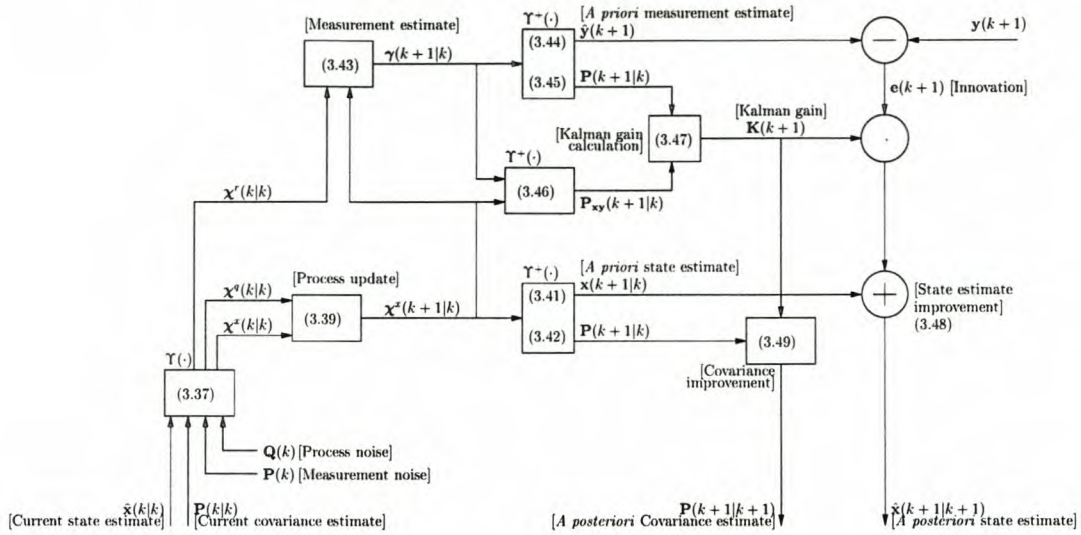


Figure 3.4: *The Unscented Kalman filter's recursive loop.*

A flow diagram depicting this process is shown in figure 3.4. At the lower left of the figure, the state and error covariance estimates together with the noise values are concatenated into the super random state variable. The UT is now applied, resulting in the three sets of sigma vectors corresponding to the state, process noise and measurement noise. The process noise and state sigma vectors are transformed by the process update equation into the *a priori* state estimate, which is combined with the measurement noise and processed by the measurement estimation to produce the estimated measurement sigma vectors.

The inverse UT is applied to the measurement estimate sigma vectors, and the resulting mean (which corresponds to the measurement estimate) is subtracted from the actual measurement to produce the innovation. The Kalman gain is also calculated, using the *a priori* state and measurement sigma points in a cross covariance calculation. The product of the Kalman gain and innovation is then added to the resulting *a priori* state estimate in the lower right of the figure. The end result is the *a posteriori* state and state error covariance estimates.

3.4 Dual-estimation

In previous work where the Kalman filter was applied to the SfM problem, the hidden state has always been implemented as a single state variable using a single Kalman filter. Recent work in the field of dual estimation by Wan et al [38] shows promise where the single monolithic state can be logically separated into two distinct parts. This means that the

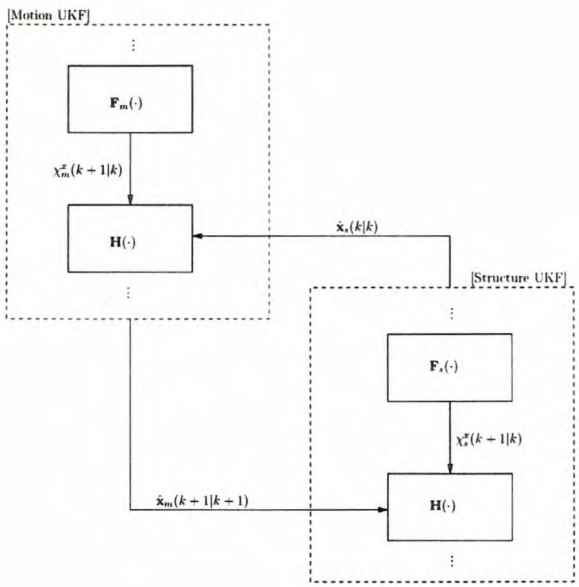


Figure 3.5: *A simplified flow diagram of the proposed SfM dual estimation strategy.*

- the dynamic *state* as well as
- the static *model*

of a system require simultaneous estimation, and it is implemented using two Kalman filters to separately estimate the state and the model.

Recall that in SfM, the state to be estimated consists of two distinct sub states:

- The motion information, which may vary dynamically and
- the structural information, which is static (for a rigid object).

We propose that the *motion* and the *structure* of the object correspond to the dynamic state and static model of a dual-estimation system.

Thus we propose the use of two separate Kalman filters. The first filter's state variables estimate the motion (rotation and translation) of the object. The second filter's state variables estimate the structure of the object.

The first (motion) filter uses the estimate of structure provided by the second filter to estimate the motion. The second (structure) filter uses the estimate of motion produced by the first filter to estimate the structure. This is depicted in figure 3.5.

The motivation for using dual-estimation are as follows:

- It is natural to remove any coupling between motion and structure, since these states are physically independent.

- Our large single state estimate, consisting of motion *and* translation, is divided into two separate states. These states are smaller, and the operations performed on the covariance matrices are faster and more accurate.

Note that separate process update functions $\mathbf{F}_s(\cdot)$ and $\mathbf{F}_m(\cdot)$ estimate structure and motion respectively. The measurement estimate function $\mathbf{H}(\cdot)$ is used unmodified in both filters. Details of the state variables, process functions and measurement functions are given in the next chapter.

3.5 Summary

The Kalman filter is a linear discrete-time estimator which minimises the estimate of the state error covariance, to recursively produce the estimated state of a dynamic system. It is computationally efficient, relying only on the current estimate and the new observations during computation, instead of requiring all previous measurements. This makes it suitable for real-time implementation on a digital computer. It has found applications in many real-world problems [32], including but not limited to the aerospace and aeronautical fields. Various discussions on the Kalman filter include introductory material by Welch and Bishop [39], as well as more advanced material by Maybeck [25], Haykin [15] and Sorenson [32].

The Extended Kalman filter is a good *approximation* of the Kalman filter for non-linear applications. The EKF has one disadvantage, in that the error covariance matrix is calculated correctly only up to the first moment. This can be seen in the difference between the state propagation and the state error covariance propagation. Where the Kalman filter propagates these two quantities in the same manner, the EKF only propagates the error covariance using a first order approximation. This occurs again during the Kalman gain calculation, where a first order approximation is used. Under severe non-linearities, this may not be sufficient.

An intuitively better approach would be to use higher order approximations of the propagation and measurement equation calculations. Such versions of the Kalman filter do exist, but their increased complexity in terms of implementation and computation prohibit their use. Examples include the Iterated Extended Kalman filter and the Iterated Linear filter smoother. Chann discusses these filters and compares their performance in [41], and finds that these enhancements do not necessarily produce better practical performance.

The UKF is a new approximation of the Kalman filter for non-linear systems [20] [18] [19] [17] [37]. Like the EKF, it reduces to the Kalman filter when applied to linear systems. However it does produce an improved approximation by using the Unscented Transform to propagate the distributions, resulting in a more accurate calculation of the *a posteriori* state error covariance estimate. Yet it is computationally no more expensive

than the EKF.

Dual-estimation is discussed, and shows promise in the field of SfM, since the structure of the object and its motion may be regarded as two separate states. It requires the use of two Kalman filters working in tandem. The dual-UKF specifically is known to be produce good results [37] and is therefore used in the proposed algorithm.

Chapter 4

The proposed algorithm

We are using a Kalman filter implementation, and we are therefore free to choose any representation of the hidden state. However, we seek the description which is simple, elegant and allows the minimum of ambiguity in the relation between the state variables and the measurement estimates. This section describes the various models and choice of state variables.

Previous implementations have used various different state variables. Some use three variables per feature point [7] [5] [41] [43], as well as variables to estimate the translation and rotation, while others only use one [16]. Quaternions are usually used to estimate rotation, combined with an estimate of rotational velocity. Translation models also differ between implementations.

4.1 Overview

Consider the SfM problem as having $3NF$ unknowns, which consists of the N points of a 3d object tracked over F frames.

Given that the object is rigid, we can define the object's coordinate system (OCS) invariant with respect to the object's rotation and translation, so that the OCS features of the object are invariant over time. Therefore, the unknown variables now consist of the unknown (but invariant) 3d points in the OCS, as well as the set of rotations and translations of the initial OCS with respect to the CCS. In our implementation the initial structure and the subsequent motion are estimated separately, and combined to produce the current object estimate during every iteration of the Kalman filter.

The object and camera coordinate systems are illustrated in relation to each other in figure 4.1. A feature point $\mathbf{p}$ in the CCS is related to its OCS coordinate $\mathbf{p}_O$ at time τ by

$$\mathbf{p}(\tau) = \mathbf{R}(\tau)\mathbf{p}_O + \boldsymbol{\varphi}(\tau), \tag{4.1}$$

where $\mathbf{R}(\tau)$ represents the 3x3 orthogonal matrix which rotates the OCS in relation to the CCS, and $\boldsymbol{\varphi}(\tau)$ is the translation vector which positions the OCS in relation to the

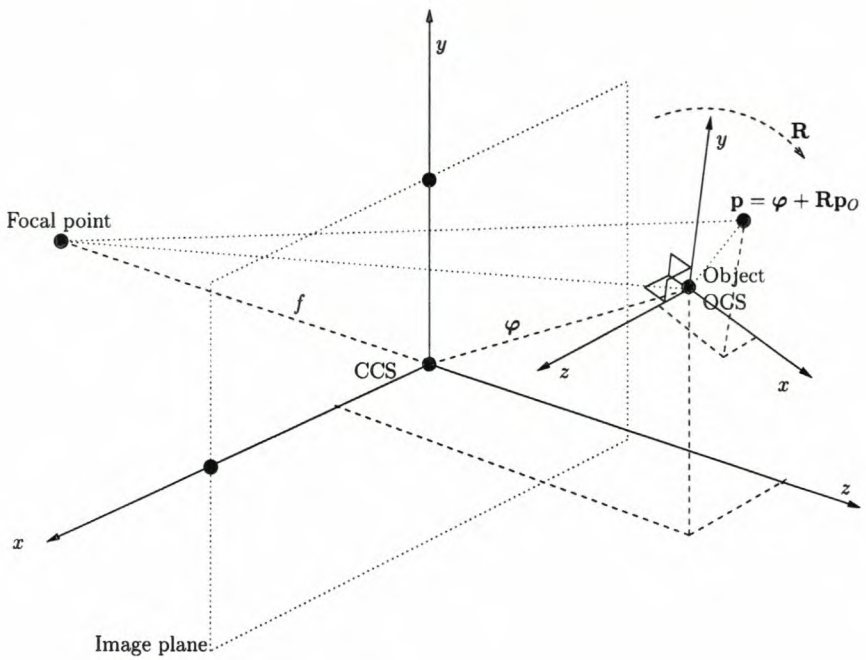


Figure 4.1: *A simplified diagram of the Camera and Object coordinate systems.*

CCS, at time τ. Note that $\mathbf{p}_O$ is not time-dependant, since the object is rigid.

In order to implement this representation of the SfM problem, we need to refine the structure model to incorporate the rigidity constraint. This is done in section 4.2. The translation and rotation models, which provides the motion model, is discussed in section 4.3

4.2 The structure model

The goal of structure estimation is to produce an estimated reconstruction of the object. This reconstruction is done in the OCS, which is invariant with respect to translation and rotation of the object, and can be achieved with only one parameter per feature due to the rigidity constraint.

Recall that only a partial reconstruction, up to a scale factor, is possible in SfM. In order to make this clear, consider the CCS point

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}. \tag{4.2}$$

From (2.3) this point projects onto the image plane of the camera as

$$\mathbf{z} = \begin{bmatrix} z_x \\ z_y \end{bmatrix} = \begin{bmatrix} p_x/(1+p_z/f) \\ p_y/(1+p_z/f) \end{bmatrix}. \tag{4.3}$$

If this point is scaled around the focal point of the camera $(0, 0, -f)$, the resulting scaled point $\hat{\mathbf{p}}$ still projects to the same image plane projection $\mathbf{z}$. Accordingly, if $\hat{\mathbf{p}}$ is the scaled version of $\mathbf{p}$,

$$\hat{\mathbf{p}} = a \begin{bmatrix} p_x + 0 \\ p_y + 0 \\ p_z + f \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -f \end{bmatrix}, \tag{4.4}$$

it projects onto the image plane as

$$\hat{\mathbf{z}} = \begin{bmatrix} ap_x/(1 + \frac{a(p_z+f)-f}{f}) \\ ap_y/(1 + \frac{a(p_z+f)-f}{f}) \end{bmatrix} \tag{4.5}$$

$$= \begin{bmatrix} afp_x/(ap_z + af) \\ afp_y/(ap_z + af) \end{bmatrix}. \tag{4.6}$$

Thus $\hat{\mathbf{z}} = \mathbf{z}$. It is therefore not possible to recover the scale a from the projection. The principle is central to the homogeneous coordinates in the projective plane of projective geometry, discussed in [12]. This allows us to drop the hat; henceforth, all variables will be scaled variables unless otherwise indicated.

We are therefore free to choose any value of a for our estimate of the object. For simplicity, we choose a to be its value that places the initial OCS origin at zero. Thus for $\varphi_z(0) = 0$, we define

$$a(\varphi_z(0) + f) - f = 0. \tag{4.7}$$

In order to complete our structure model, we introduce a variable $b(\tau)$ for each point. This value is used to fully describe the object in the OCS. It is related to the scaled z-component p_z at time τ by

$$b(\tau) = a(p_z(\tau) + f) - f. \tag{4.8}$$

From (4.8) and (4.4), we can rewrite $\mathbf{p}$ as

$$\mathbf{p}_n(\tau) = \begin{bmatrix} z_{n_x}(\tau)(1 + b_n(\tau)/f) \\ z_{n_y}(\tau)(1 + b_n(\tau)/f) \\ b_n(\tau) \end{bmatrix}. \tag{4.9}$$

This result shows that given the single parameter b_n, all of the components of $\mathbf{p}_n$ can be correctly reconstructed over time (up to the scale factor a) from the known observation $\mathbf{z}_n$ of the nth point in the set of N.

From (4.1) we can therefore write the OCS location of a point as

$$\mathbf{p}_{O_n} = \mathbf{R}^{-1}(\tau)\left(\mathbf{p}_n(\tau) - \boldsymbol{\varphi}(\tau)\right). \tag{4.10}$$

The structure can therefore be represented in terms of the initial orientation of the OCS. This means that the structure *estimate* is represented by one parameter $\beta_n(\tau)$ per feature, which is the time-varying estimate of $b(0)$ at time τ:

$$\mathbf{p}_{O_n}(\tau) = \mathbf{R}^{-1}(0) \begin{bmatrix} z_x(0)(1+\beta(\tau)/f) \\ z_y(0)(1+\beta(\tau)/f) \\ \beta(\tau) \end{bmatrix}_n - \boldsymbol{\varphi}(0). \tag{4.11}$$

We are free to choose the initial rotation estimate $\mathbf{R}(0)$ as the rotation matrix which aligns the OCS-axii with the CCS-axii, so that $\mathbf{R}(0) = \mathbf{I}$. From (4.7) we have already chosen $\varphi_z(0) = 0$. We choose (φ_x, φ_y) as the centroid $\bar{\mathbf{z}}$ of the initial image plane projections of the object points.

Therefore our estimated OCS structure is defined as

$$\mathbf{p}_{O_n}(\tau) = \begin{bmatrix} z_x(0)(1+\beta(\tau)/f) - \bar{z}_x(0) \\ z_y(0)(1+\beta(\tau)/f) - \bar{z}_y(0) \\ \beta(\tau) \end{bmatrix}_n, \tag{4.12}$$

which is fully described by

$$\mathbf{x}_s = [\beta_0,\ \beta_1,\ \beta_2,\ \ldots\ \beta_{N-1}], \tag{4.13}$$

and

$$\boldsymbol{\rho} = [\mathbf{z}_0(0),\ \mathbf{z}_1(0),\ \mathbf{z}_2(0),\ \ldots\ \mathbf{z}_{N-1}(0)], \tag{4.14}$$

where $\mathbf{x}_s$ consists of the current estimate of the initial scaled z-component of each point in the OCS, and $\boldsymbol{\rho}$ consists of the initial observations.

As a final note on this model, please note that the subsequent estimation of the object is very dependant on the accuracy of the initial observations, stored in $\boldsymbol{\rho}$. Since this information is never updated, it can have a deterious effect on the accuracy of the resulting reconstruction. However, using new $\mathbf{z}_n$ values during each iteration causes the time-propagation equation of $\beta(\tau)$ to become much more complex. This is the first trade-off in the design of the proposed algorithm.

4.3 The motion model

This section discusses the dynamic model of motion which is used by the proposed algorithm. The model describes the current rotation and translation of the OCS relative to its initial orientation in the CCS, which can be applied to the OCS structure model discussed in the previous section to produce the estimate of the current object in the CCS.

The rotation is described using a quaternion, and the translation using three values that closely follow the image plane projections as it translates.

4.3.1 The rotation sub-model

The rotation model describes the rotation of the OCS as well as its rotational velocity in relation to its initial orientation in the CCS. The rotation is represented mathematically by a normalised quaternion. Quaternions have found use in Computer Vision as early as 1983 [28]. Faugeras describes the use of quaternions to represent rotation in [13]. Wheeler and Ikeuchi describe an iterative rotation and translation estimation algorithm which uses quaterions [40]. Real-world use of quaternions is popular in Computer graphics and games, since they allow easy interpolation between two rotations [6].

A normalised, or unit, quaternion $\mathbf{q}$ has four parameters, but only three degrees of freedom, enforced by the normality constraint. It is represented by

$$\mathbf{q} = [s, \mathbf{v}] = (s, v_1, v_2, v_3) \tag{4.15}$$

$$\begin{aligned} \|\mathbf{q}\| &= \sqrt{s^2 + v_1^2 + v_2^2 + v_3^2} && (4.16)\\ &= 1. && (4.17) \end{aligned}$$

A normalised quaternion is related to the axis-and-angle representation of rotation by

$$\mathbf{q} = [\cos(\theta/2), n_1 \sin(\theta/2), n_2 \sin(\theta/2), n_3 \sin(\theta/2)], \tag{4.18}$$

so that

$$\begin{aligned} s &= \cos(\theta/2) && (4.19)\\ \mathbf{v} &= \mathbf{n}\sin(\theta/2) && (4.20)\\ (v_1, v_2, v_3) &= (n_1, n_2, n_3)\sin(\theta/2), && (4.21) \end{aligned}$$

where (n_1, n_2, n_3) is the normalised axis of rotation, and θ describes the angle of rotation about the axis. Therefore, the quaternion representation is very close to the minimal axis-and-angle rotation, and can be easily transformed into a rotation matrix. The 3×3 rotation matrix $\mathbf{R}$ can be computed from the quaternion as

$$\begin{aligned} \mathbf{R} &= R[\mathbf{q}] && (4.22)\\ &= \begin{bmatrix} s^2 + v_1^2 - v_2^2 - v_3^2 & 2(v_1v_2 + sv_3) & 2(v_1v_3 - sv_2) \\ 2(v_1v_2 - sv_3) & s^2 - v_1^2 + v_2^2 - v_3^2 & 2(v_2v_3 + sv_1) \\ 2(v_1v_3 + sv_2) & 2(v_2v_3 - sv_1) & s^2 - v_1^2 - v_2^2 + v_3^2 \end{bmatrix}. && (4.23) \end{aligned}$$

The quaternion propagates in time according to the general differential equation

$$\dot{\mathbf{q}}(\tau) = \boldsymbol{\Omega}(\boldsymbol{\omega}(\tau))\mathbf{q}(\tau), \quad \mathbf{q}(\tau_0) = \mathbf{q}_0. \tag{4.24}$$

When $0 < \boldsymbol{\omega}(\tau) - \boldsymbol{\omega}(\tau_0) << 1$ the solution of (4.24) is approximately given by

$$\mathbf{q}(\tau) = \exp[\boldsymbol{\Omega}(\boldsymbol{\omega}(\tau_0)) \cdot (\tau - \tau_0)]\mathbf{q}_0, \tag{4.25}$$

where $\boldsymbol{\Omega}$ is given by

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = 0.5 \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix}. \tag{4.26}$$

This first-order model of rotation is quite simple, and is implemented in the algorithm, since higher order models quickly become very complex. This is another example of Kalman filter design trade-off: Rather than use a complex model with little process noise, it may be better to implement a simpler model and let the Kalman filter's noise-handling ability deal with an increased process noise.

From the discussion above, it is clear that both the quaternion $\mathbf{q}$ and the rotational velocity $\boldsymbol{\omega}$ should form part of the estimation. Therefore, the state variable for rotation is composed as

$$\mathbf{x_R} = [s\ v_1\ v_2\ v_3\ \omega_1\ \omega_2\ \omega_3]. \tag{4.27}$$

Using this state variable, the rotation update equation is given by (4.25), and the estimated rotation matrix $\mathbf{R}$ is calculated with (4.23).

4.3.2 The translation sub-model

The goal of the translation sub-model is to describe the translation of the OCS in the CCS over time. This is achieved in a way that allows a change in one of the translation parameters to have the most effect on the subsequent image plane projections.

We therefore define a scale value t_z in terms of the camera's focal length and the z-component of the OCS origin as

$$t_z = 1 + \varphi_z/f. \tag{4.28}$$

We also define two image plane coordinates, t_x and t_y, that represent the pinhole projection (2.3) of the OCS origin, so that

$$t_x = \varphi_x/(1 + \varphi_z/f) \tag{4.29}$$

$$= \varphi_x/t_z \tag{4.30}$$

and

$$t_y = \varphi_y/(1 + \varphi_z/f) \tag{4.31}$$

$$= \varphi_y/t_z. \tag{4.32}$$

We can motivate our choice of coordinates as follows. Consider a CCS point, undergoing translation, at time τ:

$$\mathbf{p}(\tau) = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} \varphi_x(\tau) \\ \varphi_y(\tau) \\ \varphi_z(\tau) \end{bmatrix}. \tag{4.33}$$

Projecting onto the image plane, we find that

$$\mathbf{z}(\tau) = \frac{f}{(f + p_z + \varphi_z(\tau))} \begin{bmatrix} p_x + \varphi_x(\tau) \\ p_y + \varphi_y(\tau) \end{bmatrix}, \tag{4.34}$$

or in terms of the new variables, (4.28) and (4.30) yields:

$$\mathbf{z}(\tau) = \frac{f}{(f + p_z/t_z)} \begin{bmatrix} p_x/t_z + t_x \\ p_y/t_z + t_y \end{bmatrix}. \tag{4.35}$$

The difference in these two equations lies in the first scaling factor: it is clear that t_z is a scaling parameter, which is intuitively correct since the size of an object's projection shrinks when its distance from the camera increases. t_x and t_y describe the translations that are applied *after* the scaling, thus decoupling the two parameters from the scaling parameter t_z. As a final demonstration, consider the scaled point $\mathbf{p}'$ rewritten as

$$\mathbf{p}'(\tau) = \frac{1}{t_z} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ 0 \end{bmatrix}. \tag{4.36}$$

The time-propagation of these three variables are now derived in relation to the object's CCS velocity: Define the time-derivative $\mathbf{d}$ of $\boldsymbol{\varphi}$ as

$$\frac{d}{dt}\boldsymbol{\varphi} = \mathbf{d}. \tag{4.37}$$

Consider the time-derivative of t_z:

$$\frac{d}{dt}t_z = d_z/f, \tag{4.38}$$

which, rewritten as a backward difference, produces

$$t_z(\tau_{k+1}) = t_z(\tau_k) + \Delta\tau d_z(\tau_{k+1})/f. \tag{4.39}$$

In the case of t_x and t_y, the computation is more complicated due to the scaling factor,

$$\frac{d}{dt}t_x = \frac{d}{dt}\frac{\varphi_x(\tau)}{t_z(\tau)} \tag{4.40}$$

$$= \frac{1}{t_z}\frac{d}{dt}\varphi_x - \frac{\varphi_x}{t_z^2}\frac{d}{dt}t_z \tag{4.41}$$

$$= \frac{d_x}{t_z} - \frac{\varphi_x}{t_z^2}\frac{d_z}{f}. \tag{4.42}$$

Rewriting this equation as a backward difference and substituting (4.30) produces

$$\frac{\Delta t_x}{\Delta \tau} = \frac{d_x(\tau_{k+1})}{t_z(\tau_{k+1})} - \frac{t_x(\tau_{k+1})d_z(\tau_{k+1})}{f t_z(\tau_{k+1})}, \tag{4.43}$$

which simplifies to

$$t_x(\tau_{k+1}) = \frac{1}{1 + \frac{\Delta \tau d_z(\tau_{k+1})}{f t_z(\tau_{k+1})}} \left[t_x(\tau_k) + \Delta \tau \frac{d_x(\tau_{k+1})}{t_z(\tau_{k+1})} \right]. \tag{4.44}$$

Similarly, for $t_y(\tau_{k+1})$,

$$t_y(\tau_{k+1}) = \frac{1}{1 + \frac{\Delta \tau d_z(\tau_{k+1})}{f t_z(\tau_{k+1})}} \left[t_y(\tau_k) + \Delta \tau \frac{d_y(\tau_{k+1})}{t_z(\tau_{k+1})} \right]. \tag{4.45}$$

The $\Delta\tau$ value used in the equations above depends on the chosen time scale. For a constant frame-rate, a constant value of $\Delta\tau = 1$ therefore suffices. When the frame-rate differs from frame to frame over the sequence, the value of $\Delta\tau$ should change in proportion to the frame-rate.

We conclude with the definition of the translation state variable's composition:

$$\mathbf{x_t} = [t_x \; t_y \; t_z \; d_x \; d_y \; d_z], \tag{4.46}$$

which uses (4.44), (4.45) and (4.39) as the various time propagation functions.

4.4 Computing the *a priori* structure and motion variables

The Kalman filter calculates the *a priori* state variables during the process update step, which represent the "guess" of the state of the system during the next time-step. In this section the equations for producing these values for each model is discussed.

For structure estimation, there is no time dependency. However, the uncertainty and time evolution of the current estimate can be expressed as

$$\beta_n(\tau_{k+1}|\tau_k) = \beta_n(\tau_k) + \eta(0, Q_\mathbf{s}), \tag{4.47}$$

where $\eta(\bar{\eta}, \sigma_\eta)$ represents a Gaussian distribution with mean $\bar{\eta}$ and variance σ_η. In this case it represents zero-mean white noise with a variance $Q_\mathbf{s}$, which represents structure model noise. (Please keep in mind that while β_n is written in the form of a time-dependant value in this equation, it is defined as an time-evolving estimate of $b_n(0)$.)

Since the rotational velocity is assumed to change slowly over time, it is propagated by

$$\boldsymbol{\omega}(\tau_{k+1}|\tau_k) = \boldsymbol{\omega}(\tau_k) + \eta(0, Q_\omega), \tag{4.48}$$

In this case, $\eta(0, Q_{\boldsymbol{\omega}})$ is the zero-mean white noise which again incorporates the modelling approximation error.

From (4.25) the rotation estimation is therefore given by

$$\mathbf{q}(\tau_{k+1}|\tau_k) = \exp\left[\boldsymbol{\Omega}(\boldsymbol{\omega}(\tau_{k+1}|\tau_k)) \cdot \Delta\tau_{k+1}\right]\mathbf{q}(\tau_k) + \eta(0, Q_{\mathbf{q}}), \tag{4.49}$$

where $\eta(0, Q_{\mathbf{q}})$ represents the zero-mean white noise, which contains the first-order modelling approximation error.

The translational-velocity estimate is also assumed to change slowly over time, so that

$$\mathbf{d}(\tau_{k+1}|\tau_k) = \mathbf{d}(\tau_k) + \eta(0, Q_{\mathbf{d}}), \tag{4.50}$$

where $\eta(0, Q_{\mathbf{d}})$ represents the estimation error as well as the error in the translation model.

This in turn allows us to write the *a priori* scaling factor t_z as

$$t_z(\tau_{k+1}|\tau_k) = t_z(\tau_k) + \Delta\tau d_z(\tau_{k+1}|\tau_k)/f + \eta(0, Q_{\mathbf{t}_z}), \tag{4.51}$$

and the image plane translation estimate as

$$\begin{bmatrix} t_x(\tau_{k+1}|\tau_k) \\ t_y(\tau_{k+1}|\tau_k) \end{bmatrix} = \begin{bmatrix} \frac{1}{1+\frac{\Delta\tau d_z(\tau_{k+1}|\tau_k)}{f t_z(\tau_{k+1}|\tau_k)}} \left[t_x(\tau_k) + \Delta\tau \frac{d_x(\tau_{k+1}|\tau_k)}{t_z(\tau_{k+1}|\tau_k)}\right] \\ \frac{1}{1+\frac{\Delta\tau d_z(\tau_{k+1}|\tau_k)}{f t_z(\tau_{k+1}|\tau_k)}} \left[t_y(\tau_k) + \Delta\tau \frac{d_y(\tau_{k+1}|\tau_k)}{t_z(\tau_{k+1}|\tau_k)}\right] \end{bmatrix} + \eta(0, Q_{\mathbf{t}_{xy}}), \tag{4.52}$$

where $\eta(0, Q_{\mathbf{t}_z})$ and $\eta(0, Q_{\mathbf{t}_{xy}})$ represent the estimation error in the object's distance from the camera and its projected centre respectively.

4.5 The motion transform

The *a priori* state variables are combined to produce the estimate of the current 3d CCS state of the object at time τ_{k+1}. This calculation is called the *motion transform*, and is discussed in this section.

First, the object is reconstructed in the OCS. From (4.12), and substituting the *a priori* β_n values and the initial measurements $\boldsymbol{\rho}$ yields

$$\mathbf{p}_{O_n}(\tau_{k+1}|\tau_k) = \begin{bmatrix} z_x(0)(1 + \beta(\tau_{k+1}|\tau_k)/f) - \bar{z}_x(0) \\ z_y(0)(1 + \beta(\tau_{k+1}|\tau_k)/f) - \bar{z}_y(0) \\ \beta(\tau_{k+1}|\tau_k) \end{bmatrix}_n . \tag{4.53}$$

A rotation matrix $\mathbf{R}(\tau_{k+1}|\tau_k)$ is computed from (4.23) by substituting the *a priori* rotation estimate $\mathbf{q}(\tau_{k+1}|\tau_k)$. This is applied to the OCS reconstruction above. From (4.36) and combined with the *a priori* estimates of the scaling factor and the image plane translation to yield

$$\mathbf{p}_{\mathbf{n}}(\tau_{k+1}|\tau_k) = \frac{1}{t_z(\tau_{k+1}|\tau_k)}\mathbf{R}(\tau_{k+1}|\tau_k)\mathbf{p}_{O_n}(\tau_{k+1}|\tau_k) + \begin{bmatrix} t_x(\tau_{k+1}|\tau_k) \\ t_y(\tau_{k+1}|\tau_k) \\ 0 \end{bmatrix} . \tag{4.54}$$

4.6 The measurement estimation model

The measurement estimate applies the perspective projection to the current estimate of the 3d CCS object. This produces the 2d image plane coordinates of the current object points. Since we can measure the actual current 2d image plane coordinates of the object, we can compare these two sets of information to produce the Kalman filter's innovation value.

Taking into account measurement noise, we write $\hat{\mathbf{z}}_n(\tau_{k+1}|\tau_k)$ from the simple pinhole projection model (2.3) as

$$\hat{\mathbf{z}}_n(\tau_{k+1}|\tau_k) = \frac{1}{p_z(\tau_{k+1}|\tau_k)} \begin{bmatrix} p_x(\tau_{k+1}|\tau_k) \\ p_y(\tau_{k+1}|\tau_k) \end{bmatrix}_n + \eta(0, R) \tag{4.55}$$

$$= \frac{1}{\mathbf{r}_z \mathbf{p}_{O_n}(\tau)/t_z(\tau)} \begin{bmatrix} \mathbf{r}_x \mathbf{p}_{O_n}(\tau)/t_z(\tau) + t_x(\tau) \\ \mathbf{r}_y \mathbf{p}_{O_n}(\tau)/t_z(\tau) + t_y(\tau) \end{bmatrix}_{\tau=(\tau_{k+1}|\tau_k)} \tag{4.56}$$

$$+ \eta(0, R), \tag{4.57}$$

where $\eta(0, R)$ represents the measurement noise due to quantisation errors in the CCD of the camera.

4.7 Implementation

Previous Kalman filter based algorithms have estimated the entire unknown state of the system using one Kalman filter. If the problem is formulated as above, the problem can be clearly separated into the problem of initial (invariant) structure estimation, and the problem of translation and rotation estimation. Our proposed algorithm implements these two estimations as a dual-UKF filter, which was discussed in chapter 3.

The algorithm's flow diagram is depicted in figure 4.2. In the Kalman Process update step, the estimates of the initial structure (in the OCS) and current motion (in the CCS) is calculated. In the Measurement estimation step, the motion is applied to the structure. This aligns the estimated structure with the estimated orientation of the object. The image plane projection of each point is now calculated. During the Kalman Innovation, these estimated measurements are compared with the actual measurements.

4.8 Choice of initial conditions

The initial conditions of the algorithm include not only the state variables, but also the state error covariance matrices, and the values of the process and measurement noise.

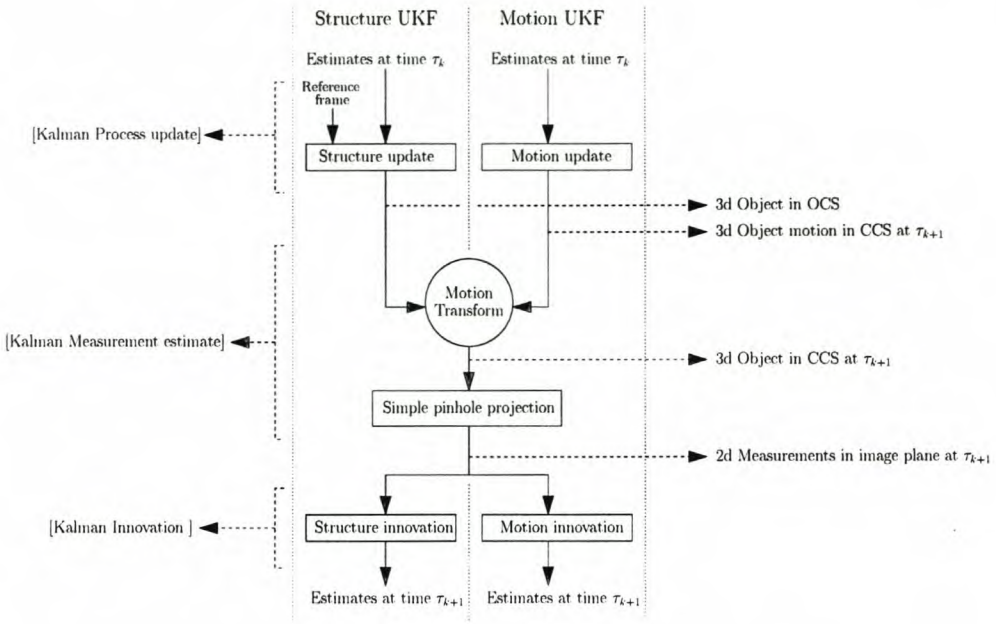


Figure 4.2: *A flow diagram of the dual estimation process.*

4.8.1 Initialising the state variables

Some of the state variables are initialised regardless of initial information about the object under observation: $\mathbf{q}$ is set to the vector $(1, 0, 0, 0)$ regardless of initial information since the initial rotation is the identity matrix. From section 4.2 we have already chosen $(t_x(0), t_y(0), t_z(0)) = (\bar{z}_x(0), \bar{z}_y(0), 1)$.

However, all the state variables require initialisation, and sensible values must be chosen if they cannot be obtained from other source(s). These include the β values, the initial rotational velocity $\boldsymbol{\omega}$ and the initial translational velocity $\mathbf{d}$. The $\boldsymbol{\omega}$ values are initialised to zero, indicating that the algorithm assumes zero rotational velocity.

When nothing is initially known about the structure of the object, $\beta_n(0) = 0$ is a good choice for each point. From 4.12 this makes the initial estimated object appear to be a flat plane parallel to the image plane.

4.8.2 Initialising state error covariances

In general it is not easy to estimate the initial error covariance since little *a priori* information may be available. Our experiments indicated that the results are not extremely dependant on a specific choice of initial state error covariance value. However, a value of $\sigma = 10^{-3}$ produced good results. Haykin [15] has more detail on the theoretical determination of state error covariance.

4.8.3 Initialising process and measurement noise values

In the previous sections, numerous references are made to the quantities $Q_{\mathbf{s}}$, $Q_{\mathbf{q}}$, Q_{ω}, $\mathbf{Q}_{\mathbf{t}_{xy}}$, $Q_{\mathbf{t}_z}$ and $Q_{\mathbf{d}}$. These are, respectively, the structure noise, rotation noise, rotational velocity noise, xy-translation noise, z-scaling noise and translational velocity noise. A suitable value for the process noise is found through experimentation to be 10^{-2}.

Measurement noise may be estimated from knowledge about the system, in our case the resolution of the CCD of the camera. We assume that our input data has a maximum of 1 pixel-width of noise on it. Our image is, for example, 640 by 480 pixels. We first scale the image so that its coordinates lie between -1 and 1 in both the x and y axii. Therefore, 1 pixel difference produces an error $\mathbf{e}$ of

$$\mathbf{e}_{640\times480} = \begin{bmatrix} \frac{1}{640} \\ \frac{1}{480} \end{bmatrix} = \begin{bmatrix} 0.0015625 \\ 0.0020833 \end{bmatrix}. \tag{4.58}$$

The maximum value of 0.002 motivates us to choose this value as the global measurement noise value.

As a final remark, please keep in mind that the above-mentioned noise values form the diagonal values on square matrices, in order to correctly represent the assumed zero-mean noises present in the system.

4.9 Summary and Conclusion

In this chapter the internals of the algorithm were discussed: the choice of state variables, their update and measurement equations, the noise parameters and covariance initialisation.

Using the rigid object assumption, it was possible to use only one degree of freedom β per feature. Moreover, these values are time invariant, and are only updated due to process noise.

The motion state variable, consisting of rotation and translation, was derived, along with the update and measurement equations. It is clear that the choice of motion state variables is very close to the observed data, allowing a simple measurement estimation function.

Chapter 5

Experimental results

In this chapter, the results of various practical experiments are discussed. These experiments include:

- Synthetic input is generated, ranging from pure translation in all three axii, to pure rotation around all 3 axii, to various combinations of translation and rotation. Accurate ground truth data is therefore available. Our algorithm and an EKF algorithm are both run on this data, and the results compared in section 5.1.
- A synthetic video sequence is generated and processed with feature tracking software. This experiment tests the influence of tracking errors on the algorithm, yet still provides us with accurate ground truths for error calculation. See section 5.2
- Real-world video-footage is taken and processed with a feature tracker. One set of this data has ground-truths available, and is used for verification. The results are discussed in sections 5.3 to 5.4.

It is shown that this filter is more stable and, in some cases, more accurate than a previous EKF implementation. It even performs well when used without any prior initial data, and is also demonstrated in a real-world situation.

5.1 Numerical simulation and comparisons against an EKF-based algorithm

Chann [41] previously tested three different Kalman SfM algorithms, and compared their output. The three algorithms were based on the Extended Kalman filter (EKF), the Iterated Extended Kalman filter and the Iterated Linear Filter Smoother respectively. The outcome of the comparison put the EKF-based algorithm ahead of the others. Accordingly, we compare our UKF-based algorithm with the EKF algorithm.

The EKF-based algorithm is taken from an implementation of Broida's algorithm, written by Robert Chann and Steven Blostein [5]. The code was kindly provided by Lin Zhao. The input data is chosen to consist of 3 different sequences. Comparison criteria are formulated, computed, and the result is discussed in the conclusion.

5.1.1 Goal

We attempt to compare the two algorithms in terms of structure as well as motion estimation accuracy. The proposed algorithm is also evaluated in terms of translation and rotation estimation as well.

5.1.2 Test sequences

The data consists of three test sequences:

1. x-translation with z-acceleration
2. xy-rotation with z-acceleration
3. xyz-translation and rotation, with a random non-linear change in motion midway through the sequence

Each test sequence was run with 5 different sets of initial information provided to each algorithm. This included

- a perfect set of initial data (referred to as 0% in the text),
- three sets with 20%, 50% and 100% error respectively and
- a set with zero initial data (referred to as -1).

A Monte-Carlo run, consisting of one hundred iterations, was done with each test and each initial data set, resulting in fifteen sets of results.

5.1.3 Comparison criteria

The comparison in [41] is relatively easily achieved, since all of the algorithms implement the same state variable structure. In our case, however, the comparison is not so simple: since our algorithm does not implement the same state variables as the EKF algorithm, a direct state variable comparison is not possible. Instead, we compare the actual reconstructions. To do this, we track N features per frame, over F frames, and compute the following criteria for each reconstruction:

1. 2d feature error.

   Measure the 2d RMS error between the estimated 2d observations and the actual 2d observations per frame. This is an easy calculation, since we have the image plane coordinates of the actual object even if we have no 3d data.

   Therefore,

$$e_d = \sqrt{\frac{1}{F}\sum_{f=1}^{F}\frac{1}{N}\sum_{n=1}^{N}(d_{f,n})^2} \tag{5.1}$$

   is computed, where

$$d_{f,n} = \sqrt{(\hat{\mathbf{p}}_{n_x} - \mathbf{p}_{n_x})^2 + (\hat{\mathbf{p}}_{n_y} - \mathbf{p}_{n_y})^2} \quad \Big|_f, \tag{5.2}$$

   is the distance between the nth point's estimated and actual observations in the fth frame.

   If the e_d value is small, the algorithm's estimated state produces accurate observation estimates.

2. Normalised scale error variance

   In section 4.2 it was shown that if the estimated object's structure is scaled by a joint scale factor a, it is considered a sufficiently correct estimate. Conversely, if the scale factor differs between points, so that $a_i \neq a_j$ for a set $j, i \in N$, the estimation is not correct anymore. However, some estimations may be more accurate than others. A method of calculating the estimation quality is required to compare the scale factors for all points. We propose the variance in scale factors as the quality measure, since the smaller this value, the more similar the scaling factors and the higher the quality:

   Measure the variance of the scaled z-scaling factors per frame, and calculate the RMS value of these variances for the sequence. Therefore,

$$e_s = \sqrt{\frac{1}{F}\sum_{f=1}^{F}\frac{1}{N}\sum_{n=1}^{N}\left(1 - \frac{s_{n,f}}{\bar{s}_f}\right)^2} \tag{5.3}$$

   is computed, where

$$s_{f,n} = \frac{(\mathbf{P}_z)_n}{(\hat{\mathbf{P}}_z)_n} \quad \Big|_f, \tag{5.4}$$

   and

$$\bar{s}_f = \frac{1}{N}\sum_{n=1}^{N}\frac{(\mathbf{P}_z)_n}{(\hat{\mathbf{P}}_z)_n} \quad \Big|_f. \tag{5.5}$$

   If the e_s value is small, then the algorithm's estimation is almost an exact scaled replica of the object.

It is essential to realise that it is the *combination* of these two criteria which determines how accurate the 3d reconstruction is. If e_d and e_s are *both* small, then we can be certain that the reconstruction is accurate[1].

A subtle error might pass by unnoticed if these two values are the only available comparison data: since variance is by definition always positive, the invalid case where the scaling is negative (the object is estimated to be behind the camera[2]) can not be detected from these criteria alone. Therefore, each result must be verified visually to detect this possible error.

5.1.4 Results

To facilitate the discussion, our algorithm will be referred to as the "UKF algorithm", while the EKF algorithm will be referred to as the "Broida algorithm". In figures 5.1 to 5.3 the computed e_d and e_s values can be seen for each of the fifteen test runs, grouped by sequence.

From the results, the following can be deducted:

- In the first sequence, the UKF outperforms the Broida algorithm across the board, producing comparable if not better 2d tracking e_d values (especially with less initial information), as well as much better scaling e_s values. Another noticeable result is the absence of a result in the case of zero initial data for the Broida algorithm. This is because the algorithm became unstable during a number of test runs.

- For the second sequence, results are mixed, as the Broida algorithm outperforms the UKF in e_d values. The UKF does beat the Broida in e_s values when some initial data is present. However, the Broida algorithm beats the UKF in the case of zero initial information.

- The third sequence is better handled by the UKF algorithm, where it outperforms the Broida algorithm in all cases. Again, the Broida algorithm became unstable during some runs with a particular set of initial data.

The results of the motion estimation for all three experiments, with an initial data set corrupted with 50% noise, are shown in figures 5.4 to 5.9.

Note that the t_x axis and t_y axis measure the image width and height respectively, scaled to $[-1, 1]$. The t_z axis measures the actual normalised depth.

---

[1] Again, only up to a scale factor

[2] Although this is a *physically* impossible situation, it is a valid *mathematical* situation, given the camera model (2.3).

The UKF algorithm produces noticeably better rotation estimation results. The UKF's translation estimates are also better, especially in the third sequence after the big discontinuity.

5.1.5 Conclusion

On a diet of pure synthetic data, our algorithm based on the dual-UKF excels, and easily outperforms the algorithm based on a single EKF. The EKF only produces better results in certain cases.

On closer inspection, it also becomes apparent that the e_d values produced by the UKF algorithm can be as high as the EKF algorithm's. This is a direct result of the structure model, which uses the initial measurements throughout the whole sequence, thereby propagating the error in those measurements.

However, the UKF algorithm does produce promising results, especially in the cases where no initial information is available to the algorithms, and when big motion discontinuities occur.

5.2 A synthetic video sequence

A synthetic video sequence, generated with the free ray-tracing engine Povray [2], is processed with feature detection and tracking software to produce the frame-by-frame observation data. This data is then fed to our SfM algorithm.

5.2.1 Goal

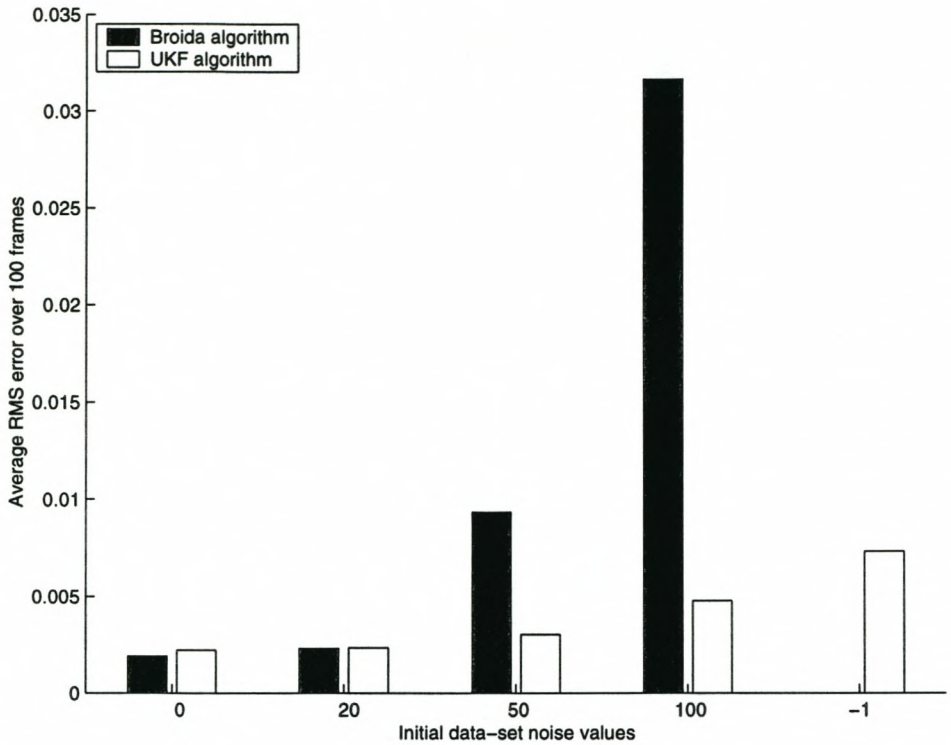
In this experiment, real quantisation and tracking errors are better approximated while still retaining complete ground truth data. Therefore this experiment forms an intermediate step between pure synthetic data and real data.

5.2.2 Generated input data

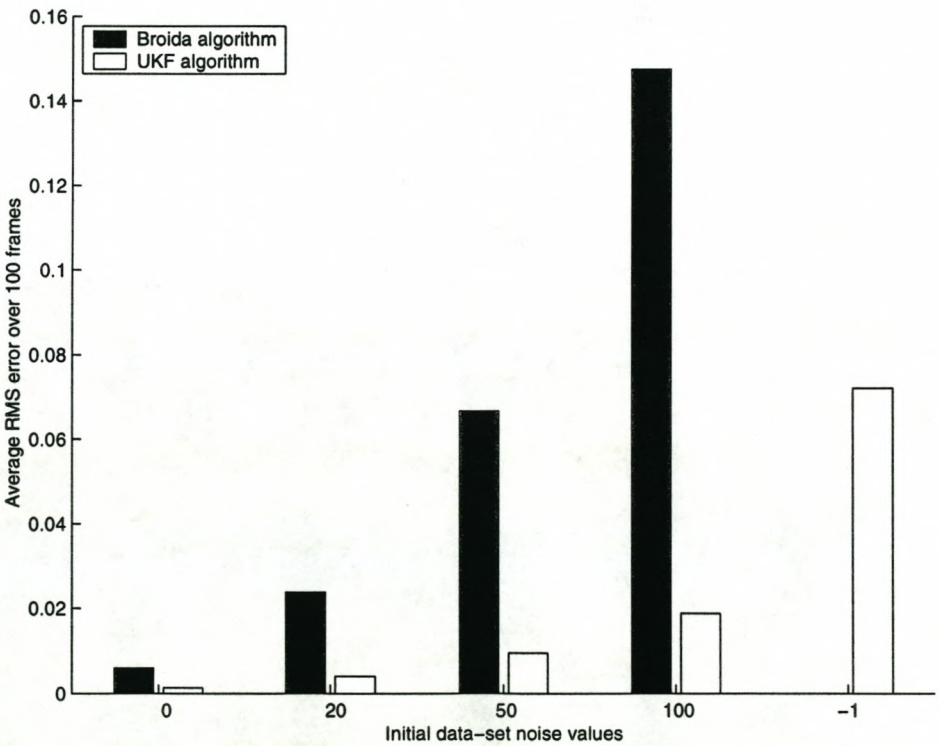
50 frames of a scene is generated of a cube which spins around the y axis. The scene is rendered in Povray, a freeware ray-tracing engine [2], and processed using a simple Kanade-Lucas-Tomasi feature detection and tracking program which uses the software library written by Stan Birchfield [4]. 6 selected frames from the sequence is shown in figure 5.10.

Our algorithm is run on the data, with no prior information of the object's structure or motion.

Ground truth data is calculated using a Matlab script implementing the Povray information. This produces the 3d locations of the 8 corners of the cube over time. However,

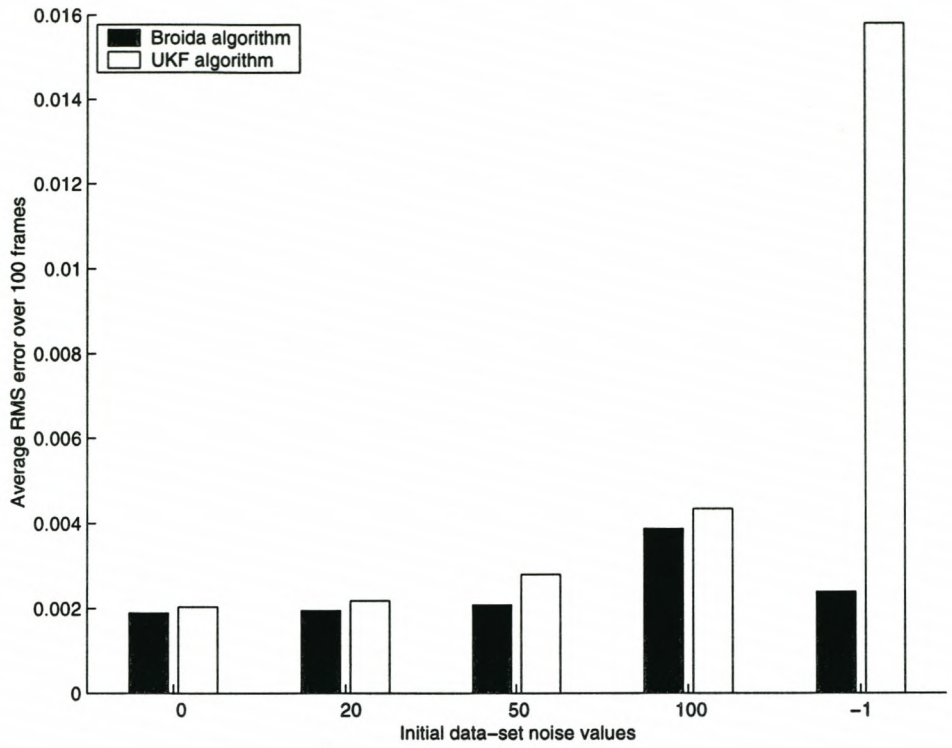


(a) e_d values for the "*x-trans*" run

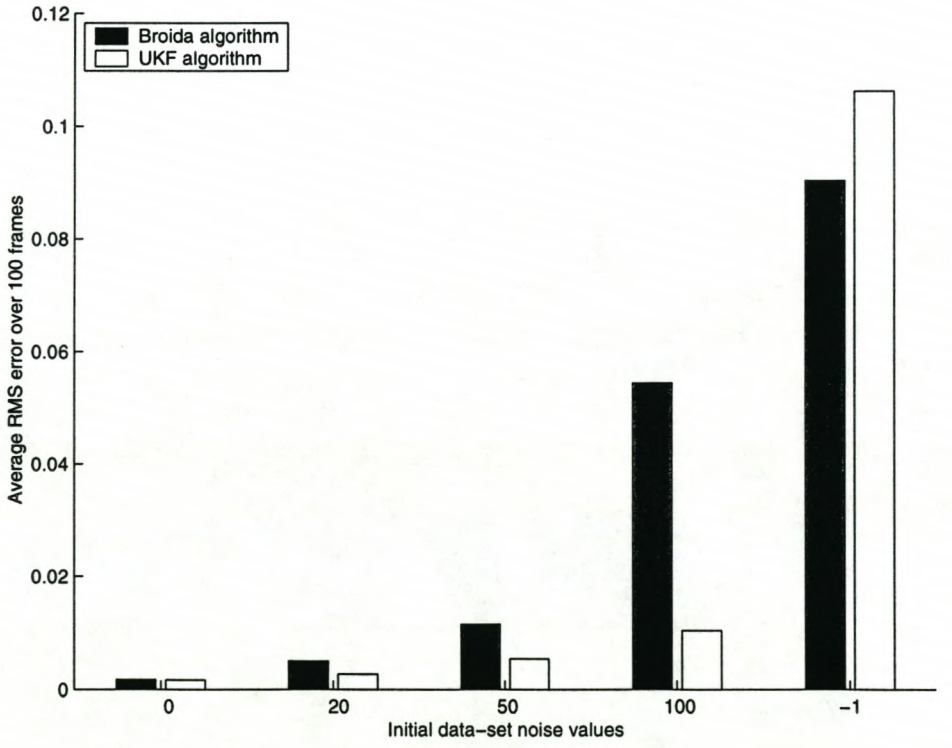


(b) e_s values for the "*x-trans*" run

Figure 5.1: *Results of the "x-trans" run*

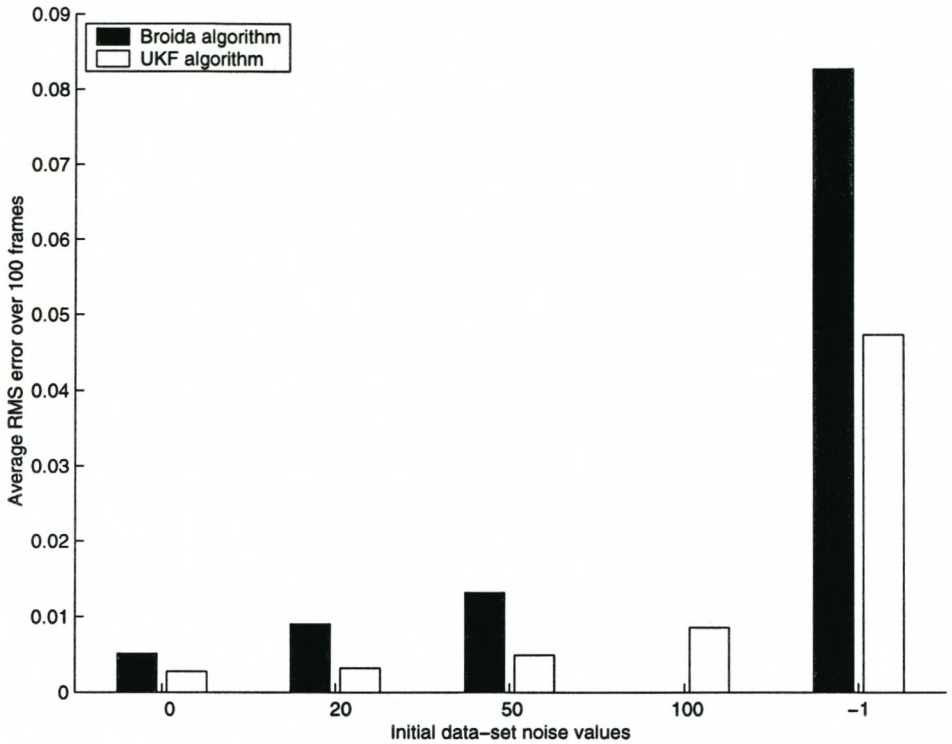


(a) e_d values for the "*z-with-rot*" run

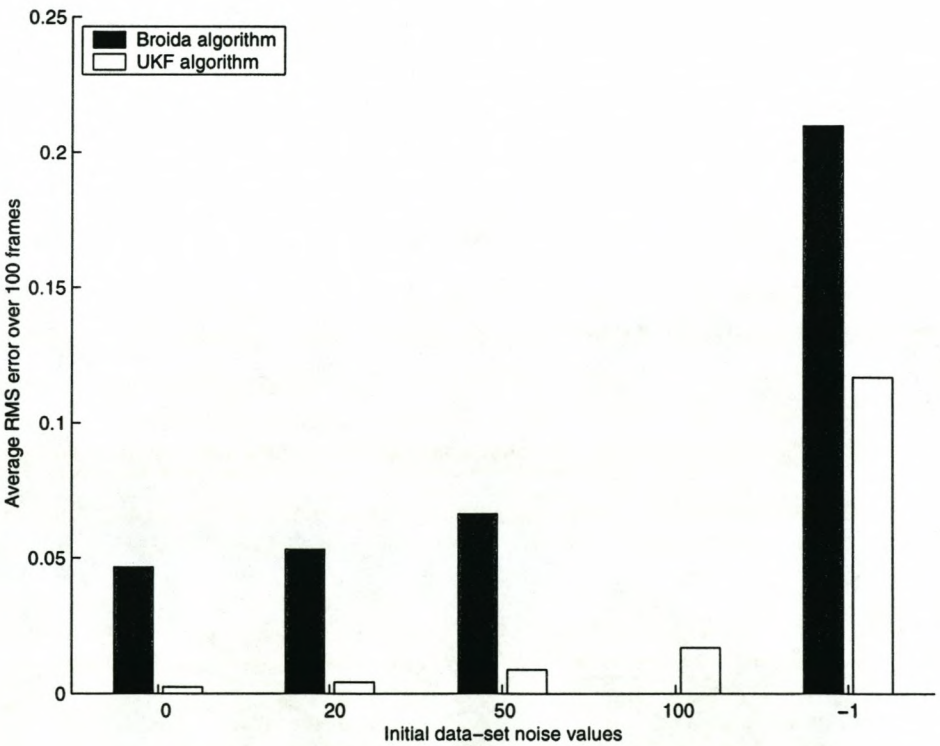


(b) e_s values for the "*z-with-rot*" run

Figure 5.2: *Results of the "z-with-rot" run*

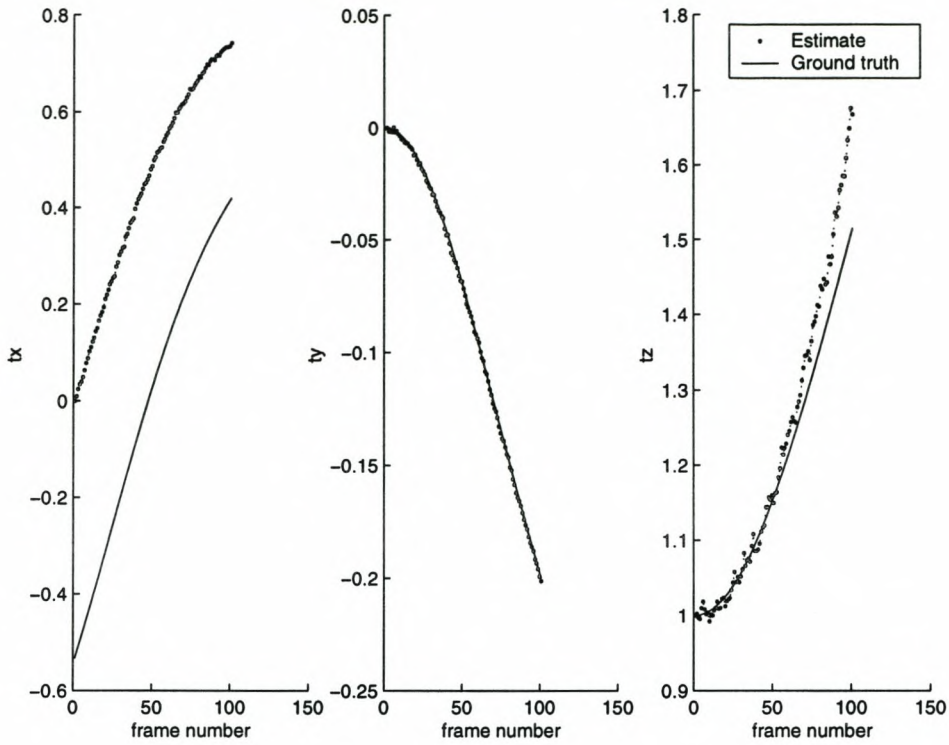


(a) e_d values for the "*broida-et-all*" run

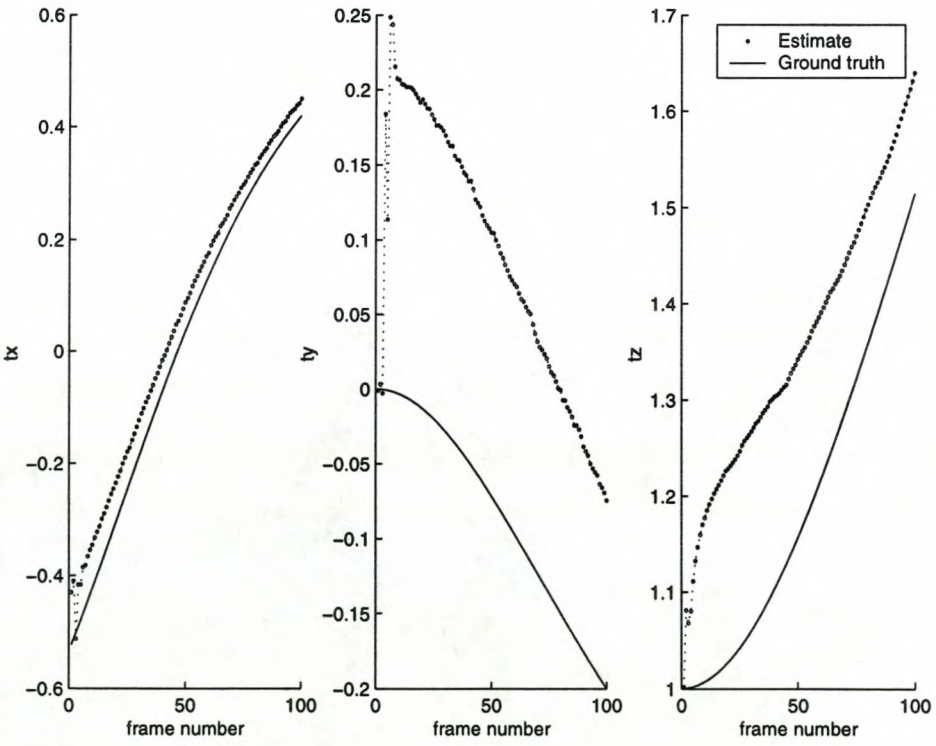


(b) e_s values for the "*broida-et-all*" run

Figure 5.3: *Results of the "broida-et-al" run*

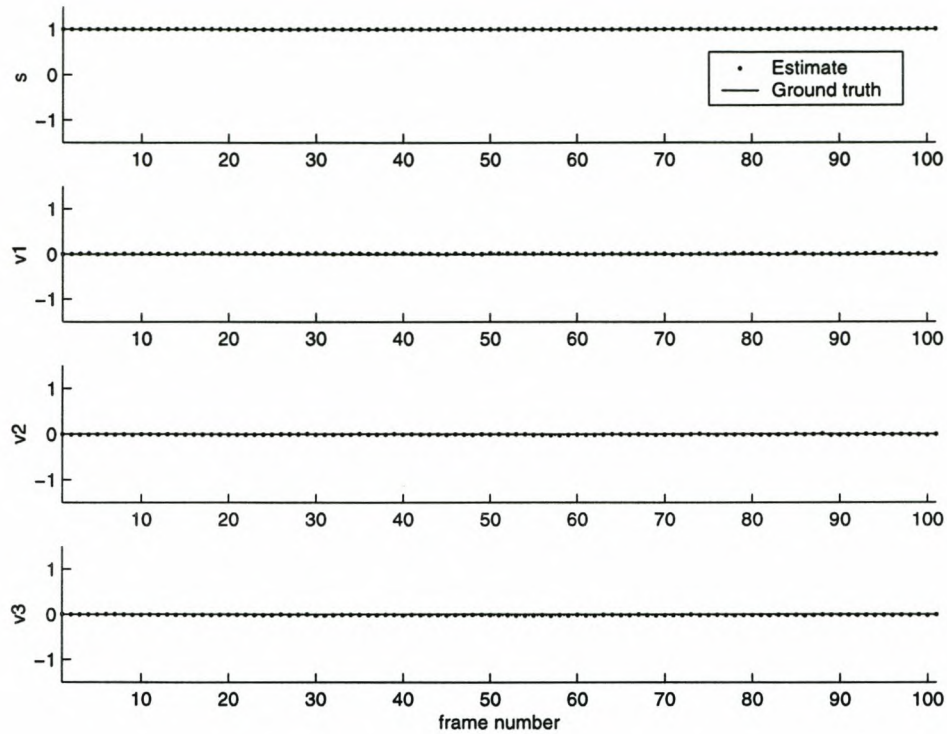


(a) UKF results

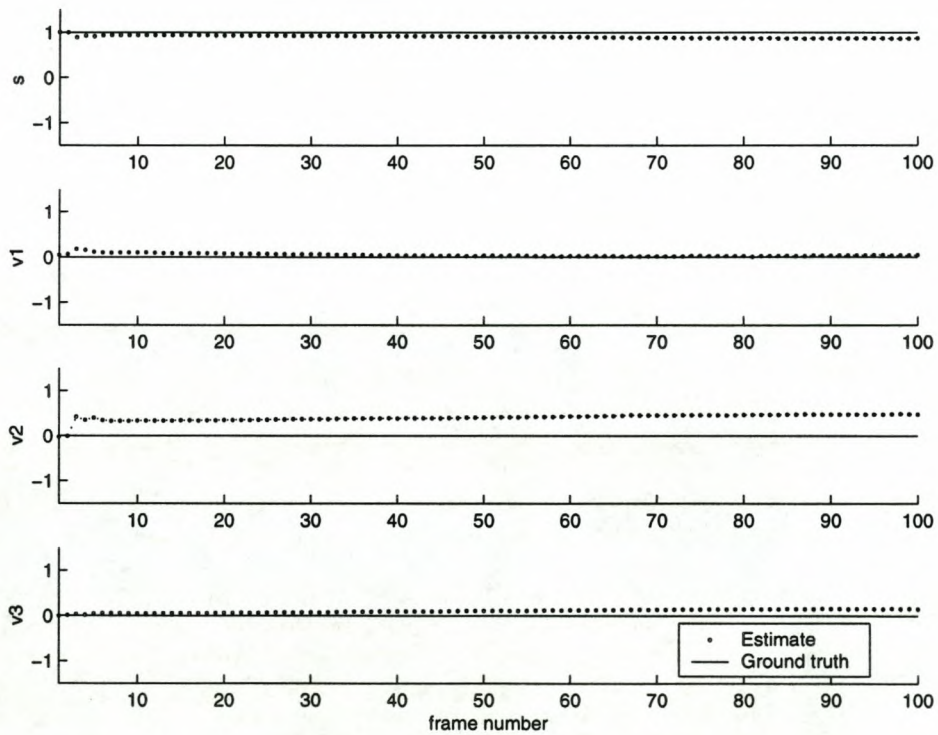


(b) Broida results

Figure 5.4: "x-trans" *translation tracking results during the 50% noise level run.*

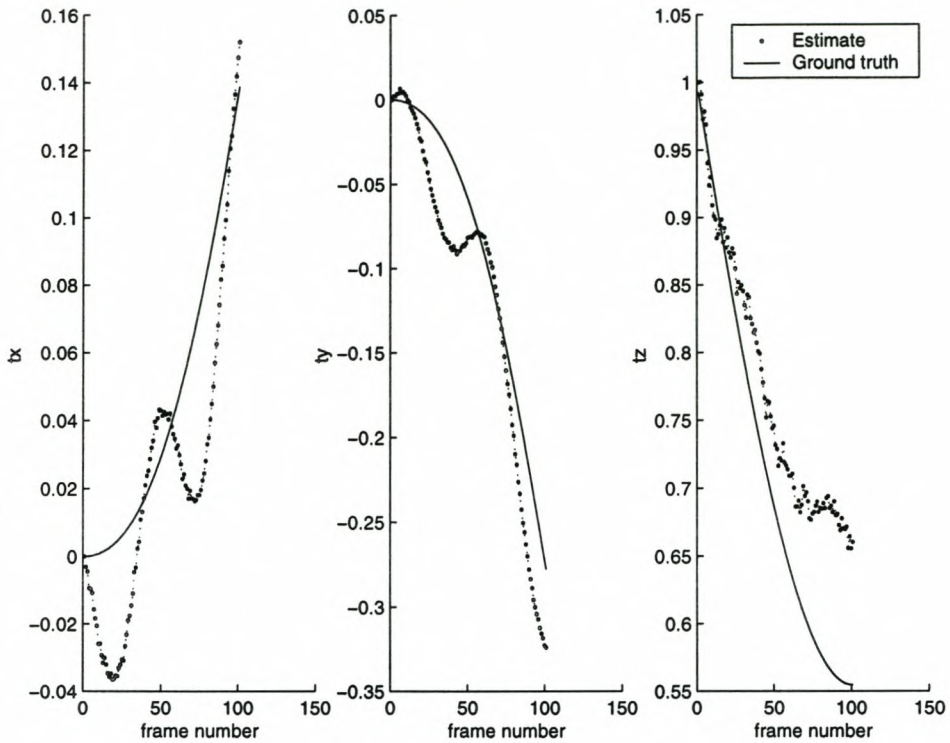


(a) UKF results

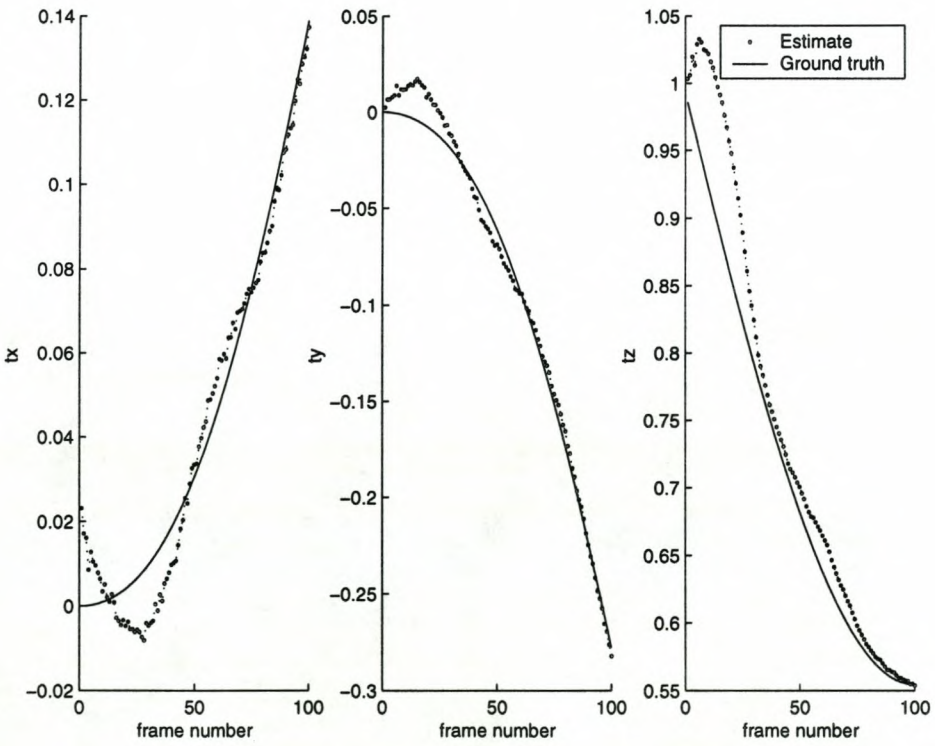


(b) Broida results

Figure 5.5: "x-trans" *rotation tracking results during the 50% noise level run.*

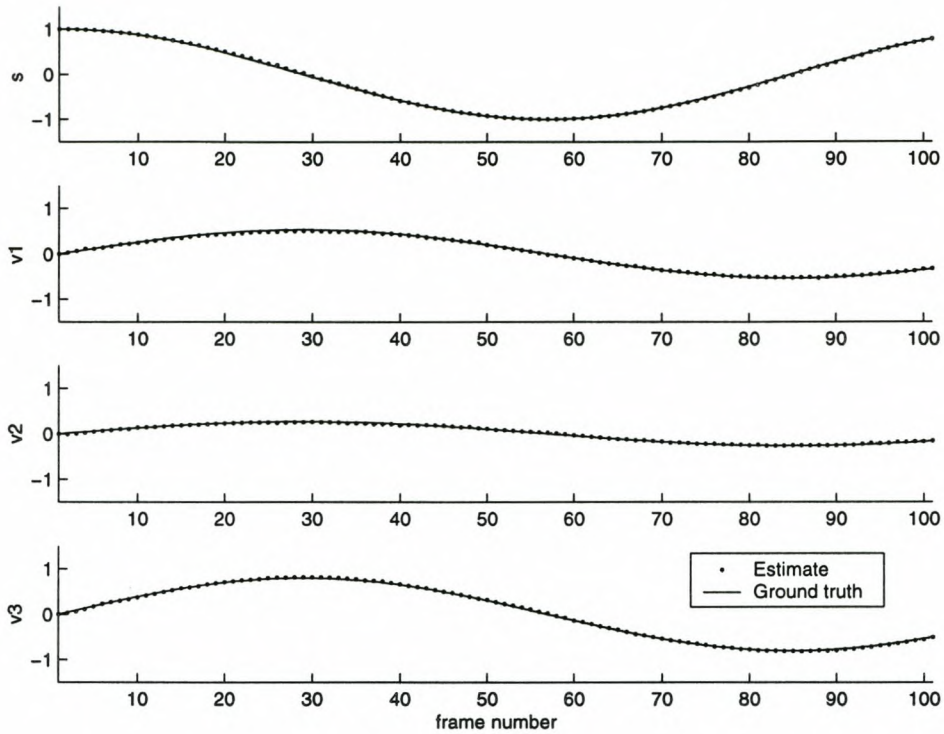


(a) UKF results

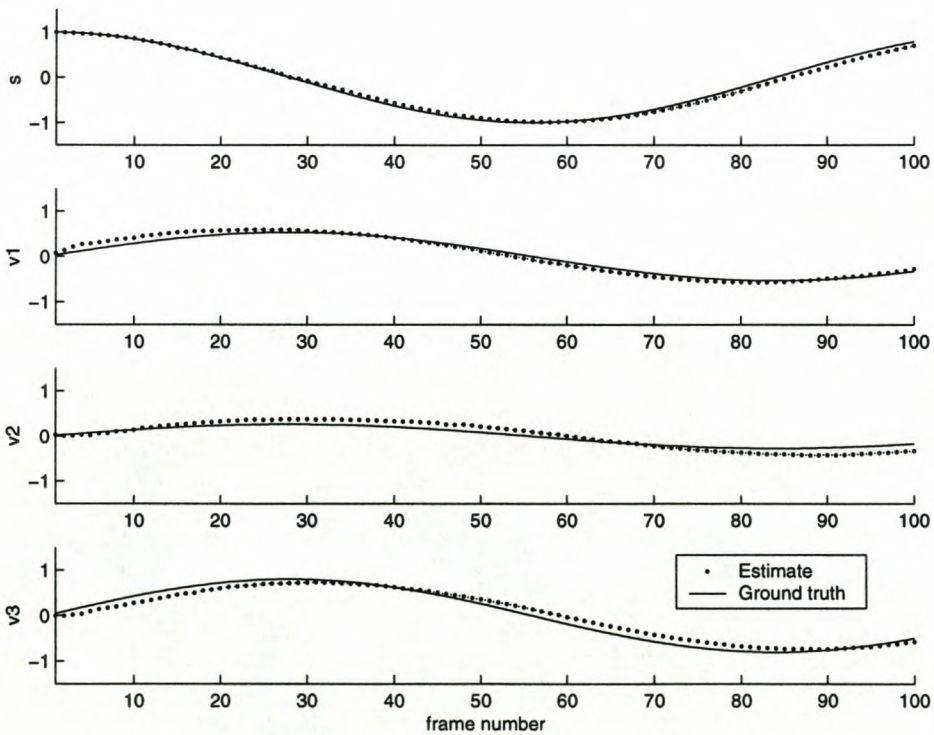


(b) Broida results

Figure 5.6: "z-with-rot" *translation tracking results during the 50% noise level run.*

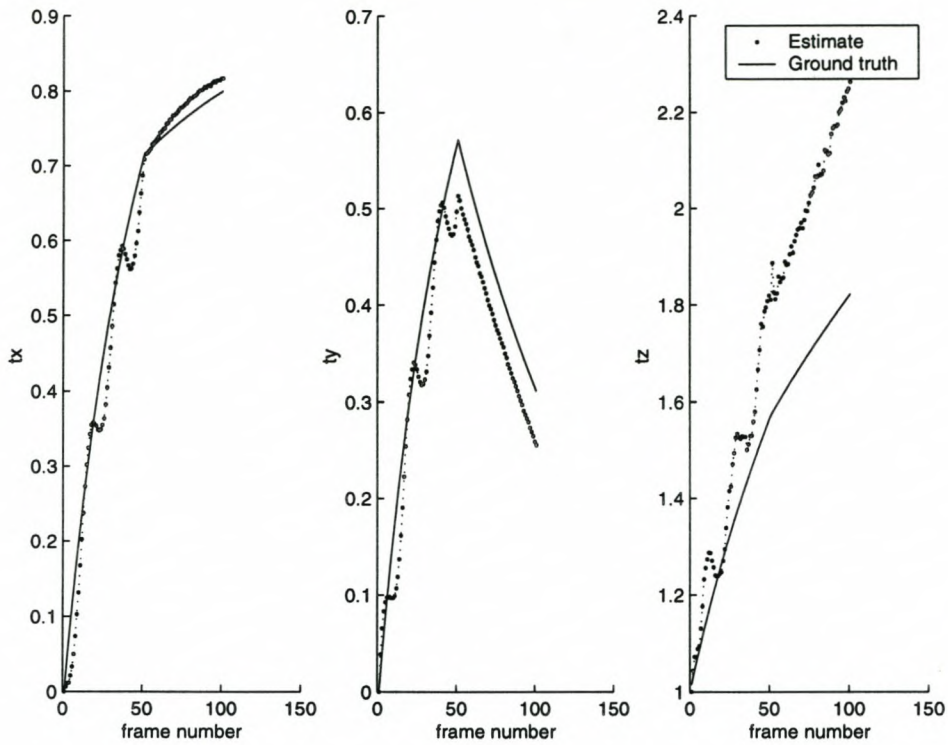


(a) UKF results

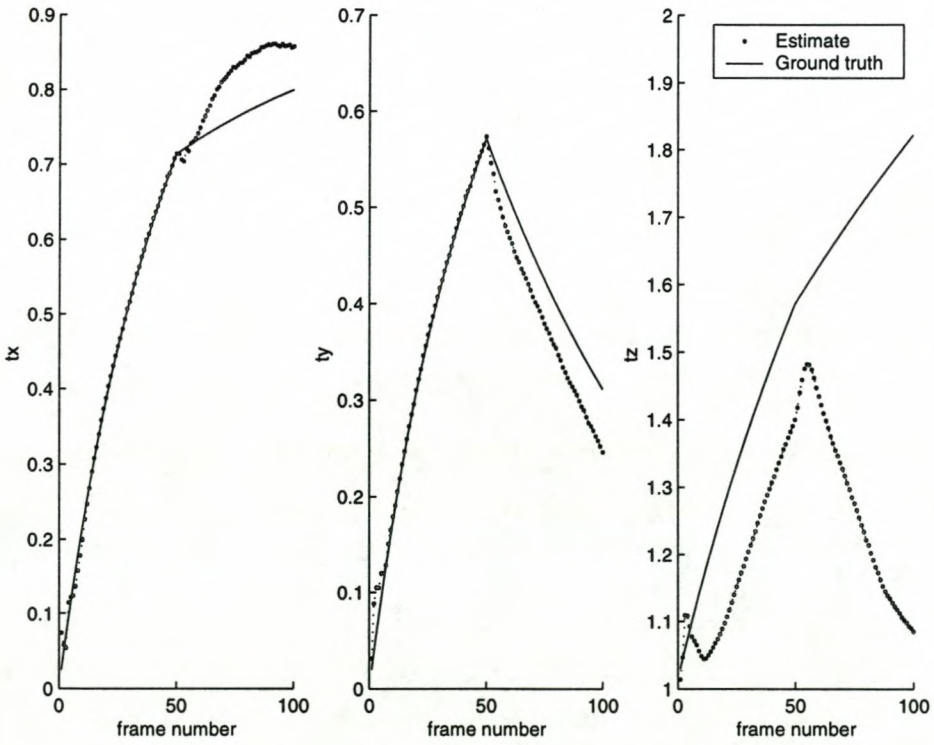


(b) Broida results

Figure 5.7: "z-with-rot" *rotation tracking results during the 50% noise level run.*

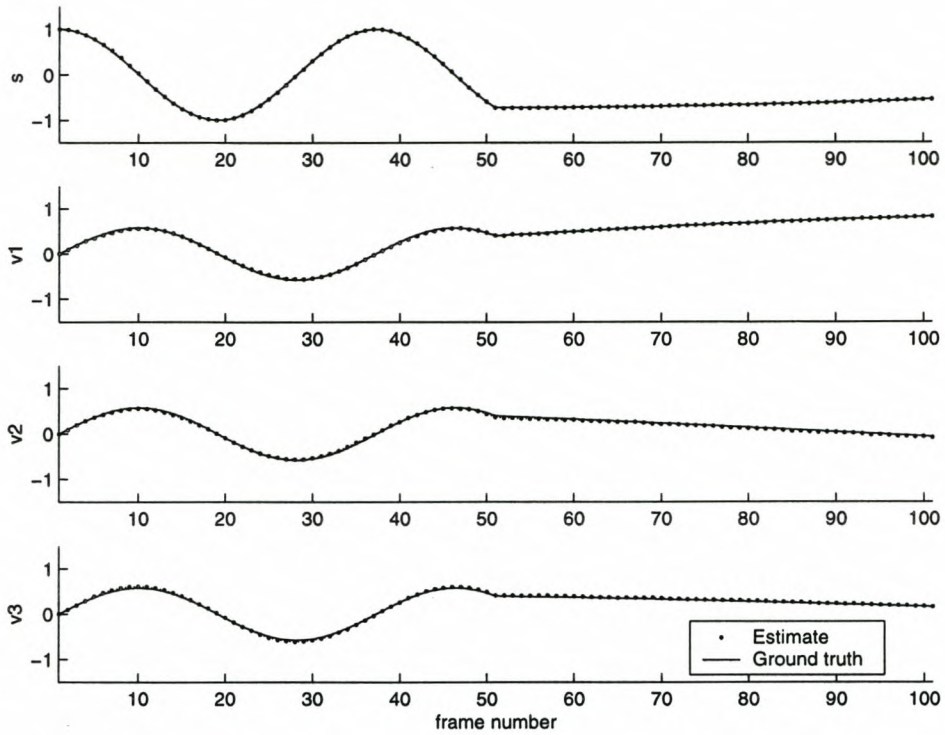


(a) UKF results

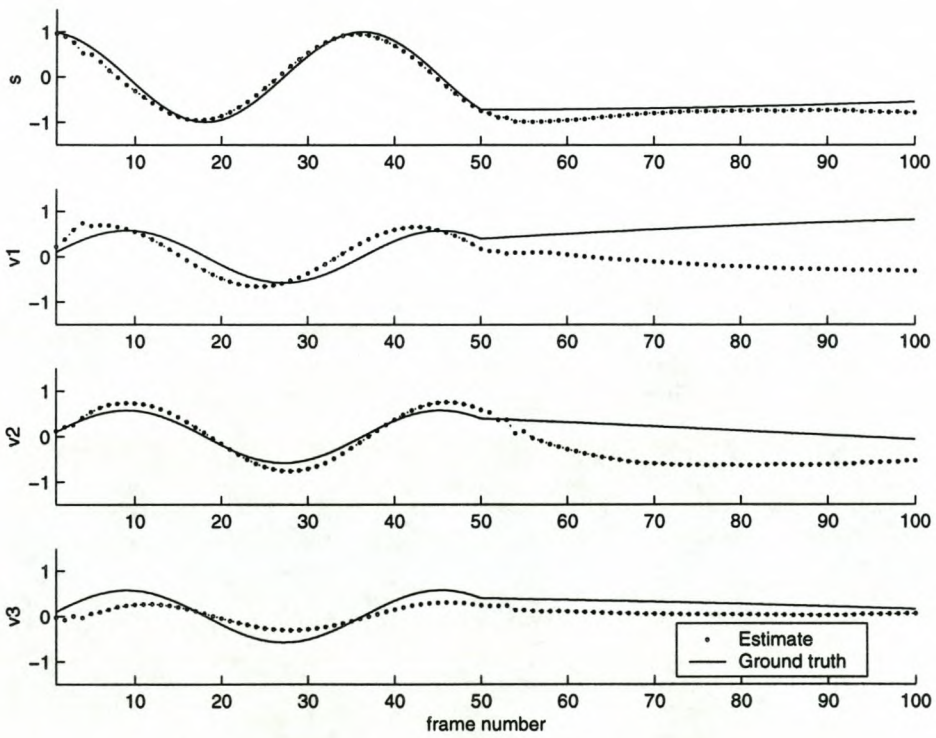


(b) Broida results

Figure 5.8: "broida-et-al" *translation tracking results during the 50% noise level run.*



(a) UKF results



(b) Broida results

Figure 5.9: "broida-et-al" *rotation tracking results during the 50% noise level run.*

the feature tracking software does not actually select and track the defined corner-points of the object. The set of estimated points and ground-truth points which correspond the most were therefore selected by hand to form the subset against which the e_s values were calculated.

5.2.3 Results

The translation and rotation estimates over time are shown in figure 5.11. It reports an object moving from right to left, while spinning around the y-axis, which corresponds to the synthetic object.

The e_d and e_s values are again calculated to determine the accuracy of the reconstruction. In this case, the ground truth 3d values for the actual object points are computed using a Matlab script, based on the Povray script:

$$e_d = 0.0215 \tag{5.6}$$

$$e_s = 0.0652 \tag{5.7}$$

This result is similar to the results achieved in the purely synthetic first experiment, where the algorithm is provided with no initial information (as is the case here).

5.2.4 Conclusion

This experiment accurately measures the quality of the output of the algorithm in an environment that closely approximates reality, since the type of measurements and the measurement noise levels are realistic. The algorithm also performs the estimation without any prior knowledge of the system.

The algorithm's output is of expected quality, and is a good estimate of the actual observed object.

5.3 The spinning radio sequence

A stereoscopic video sequence is taken of a portable radio. These sequences are processed twice: once with stereovision software programmed by Brian O'Kennedy [27], and once with our proposed algorithm. Therefore, we have accurate ground truth data with which to compare our results.

5.3.1 Goal

The goal of this experiment is to provide the algorithm with actual real-world data, which has been processed with the feature-tracking software. The accuracy of the reconstruction is computed using the available ground-truth data.

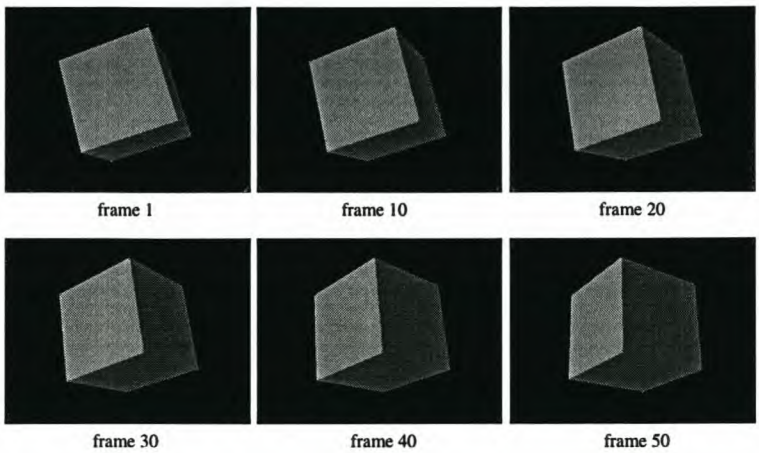


Figure 5.10: *A sample of frames from the synthetic cube sequence.*

5.3.2 Input data

An portable radio is placed on a controllable turn-table observed by two video-cameras connected to a PC running frame grabbing software. The two cameras are calibrated using a set of still pictures. Accurate intrinsic and extrinsic parameters for this setup are produced using calibration software written by O'Kennedy [27]. These cameras are then trained on the radio while it executes a controlled rotation to produce two image sequences. These sequences are also processed with O'Kennedy's software, to produce accurate ground truth data.

A sample of four stereo frames from the sequence used are shown in figures 5.12.

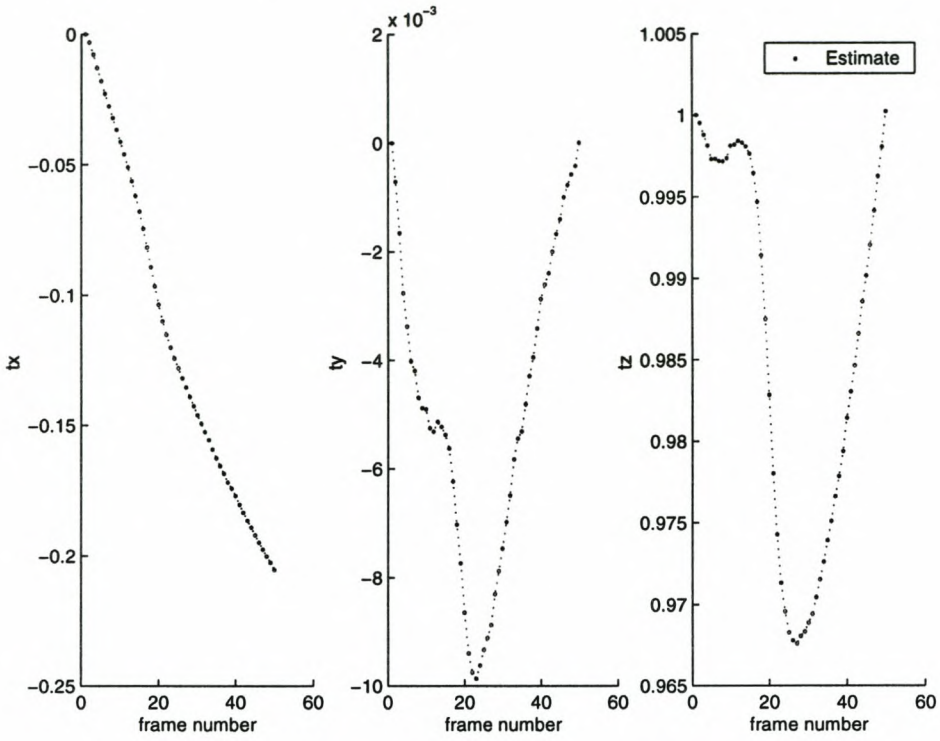
5.3.3 Results

An $e_d = 0.02966$ value is calculated for the whole sequence, providing an estimate of the measurement estimation accuracy for this experiment. e_d values of the same order are produced in the (first) synthetic experiment, for the "broida_et_al" sequence and zero initial data.

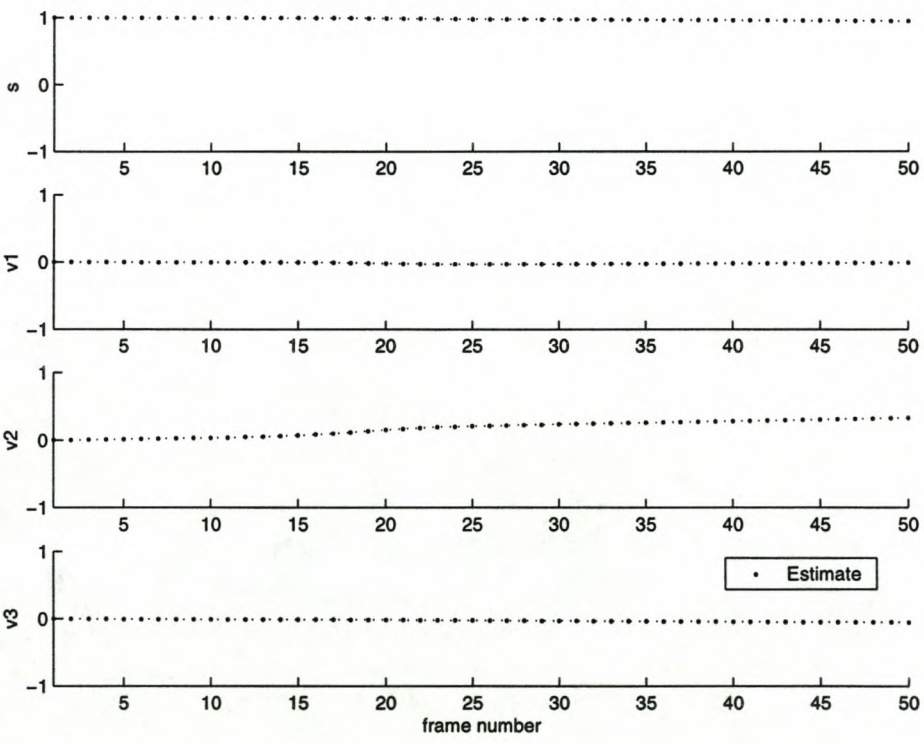
Also, an $e_s = 0.05365$ value is also calculated. This is done by comparing the final structure estimation with the reference ground truth obtained by the stereo camera calibration. Some difficulty was encountered in finding a large set of features which could be reliably tracked throughout the left sequence and the right sequence. However, we were able to track a dynamic subset of the left sequence's features in the right sequence.

The motion tracking results for the radio sequence can be seen in figures 5.13 and 5.14. As no motion information could be reliably produced, these are only provided to give an impression of the operation of the algorithm.

The estimated structure and motion of the radio is combined in order to construct a sequence of Povray scripts. These scripts describe each point with a sphere, and the



Translation tracking of the synthetic cube sequence



Rotation tracking of the synthetic cube sequence

Figure 5.11: *The translation (top) and rotation (bottom) tracking of the synthetic cube sequence.*

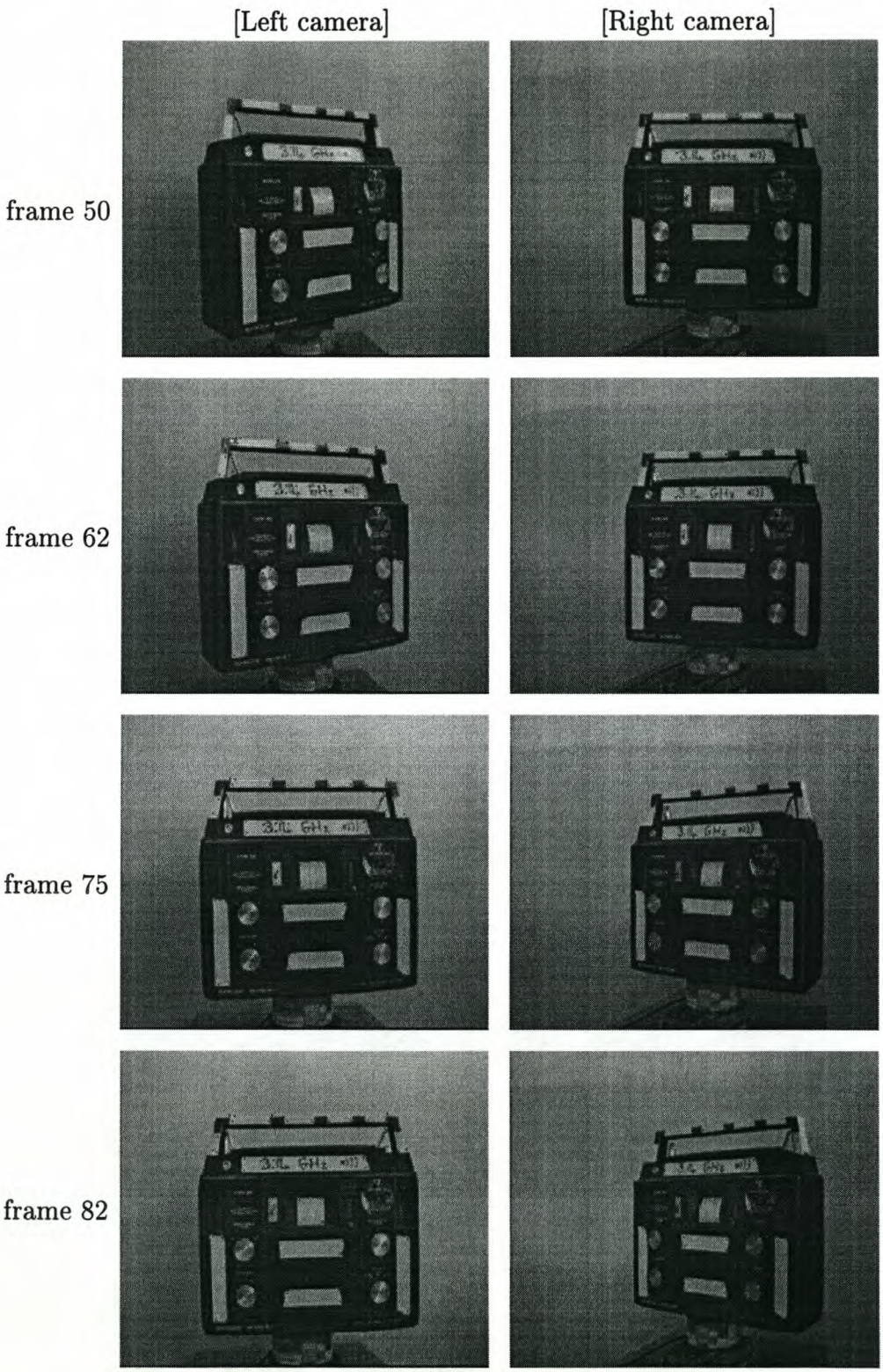


Figure 5.12: *Four frame-sets from the stereo radio sequence*

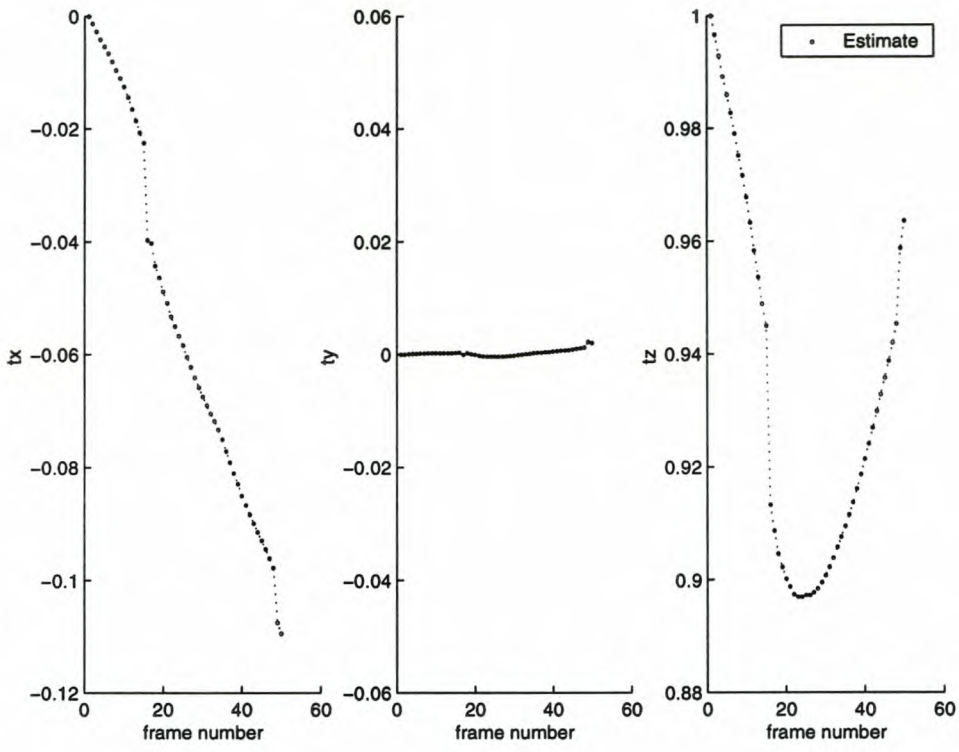


Figure 5.13: *Translation tracking of the spinning radio*

estimated motion of the object is applied to this sphere-cloud over time. These scripts are used to render the estimated object over time in relation to the left camera. A subset of the rendered sequence can be seen in figure 5.15, next to the left camera's frames. No user intervention is required for this process.

5.3.4 Conclusion

The e_d and e_s values can be placed in perspective by noting that the algorithm was not provided with *any* initialisation data. Nevertheless, it performs well, even under extreme levels of input error/noise, as is the case in this experiment.

From figure 5.13 the translation estimation seems sound, since it is evident from the video sequence that the radio does translate horizontally. Also, the rotation estimate shown in 5.14 equates to a steady rotation around the y-axis, which is known to be correct.

Therefore, from the error-calculations and the subjective judgement of the motion estimation, it is concluded that the algorithm performed very well on data produced by feature tracking software, processing real video data. This experiment demonstrates that the algorithm can succesfully operate *un-initialised* in a real world environment.

Figure 5.15 demonstrates the use of the algorithm to generate an *augmented reality*, which is the coupling of computer-generated graphics with 3d information of a image

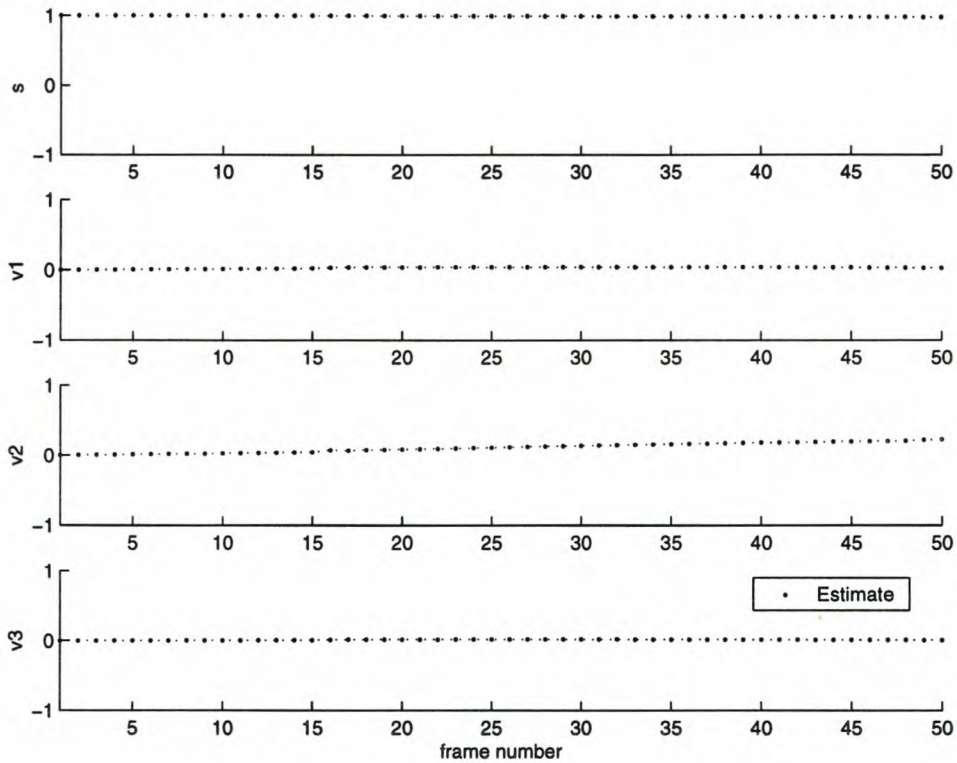


Figure 5.14: *Rotation tracking of the spinning radio*

sequence, in order to produce the illusion that the computer graphics are part of the filmed scene. This aspect of SfM has enormous commercial potential in the field of computer-generated special effects in motion pictures.

5.4 The artichoke sequence

An uncalibrated sequence, found on the Internet, is pre-processed with the feature detection/tracking software, and fed to the proposed algorithm.

5.4.1 Goal

Since no ground-truth data is available, this experiment is run only as an interesting example and the only conclusion which may be reached in this case is a subjective one.

5.4.2 Input data

The observed scene consists of a soft toy, an artichoke placed on a cylinder, and a miniature toy road-sign, with the camera panning from left to right in front of it. It is highly detailed, which makes it ideal for the feature tracking software. The camera translates from left to right in front of the scene, with no rotation.

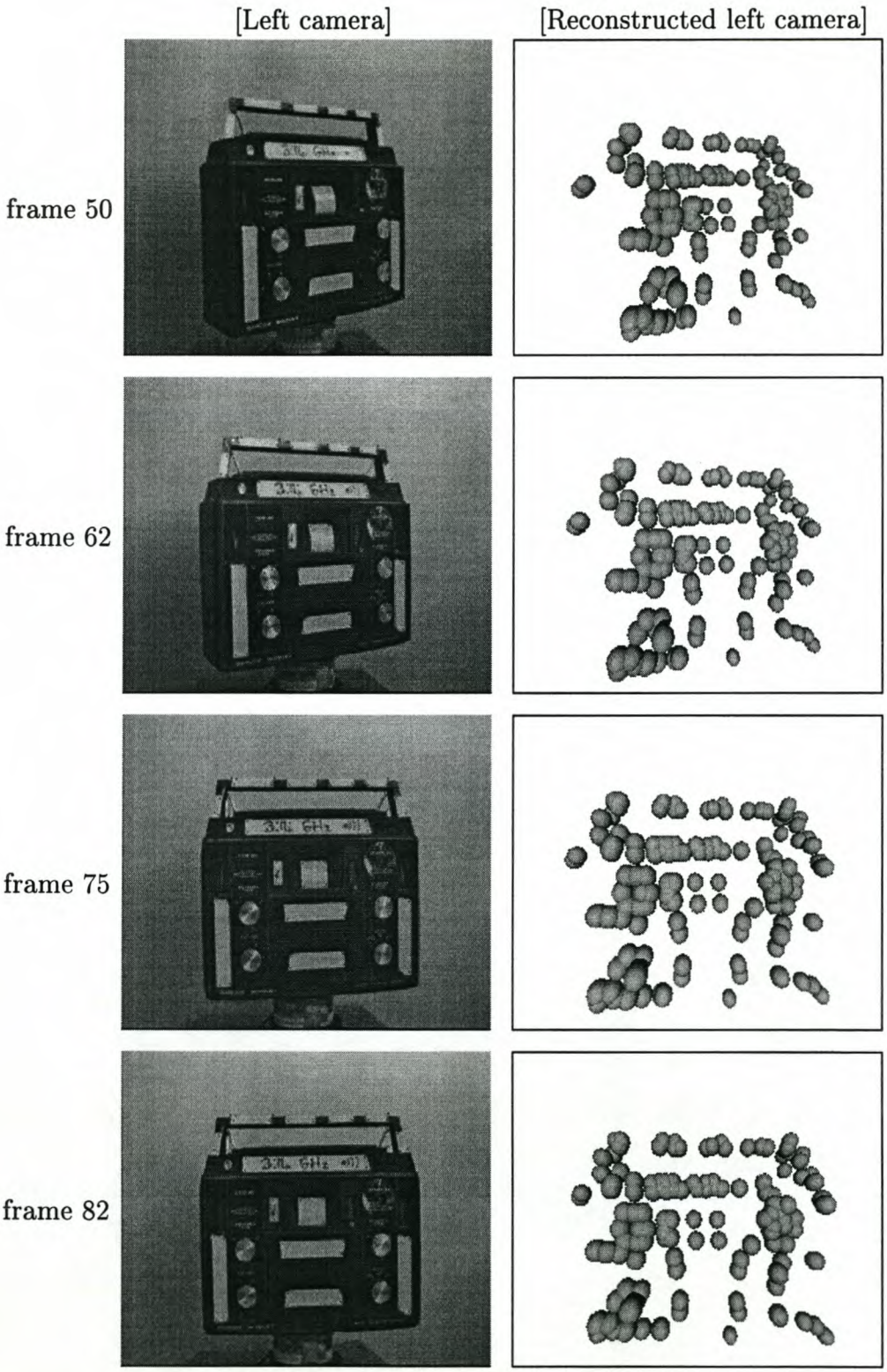


Figure 5.15: *Four frames from the left camera and the Povray-rendered reconstruction*

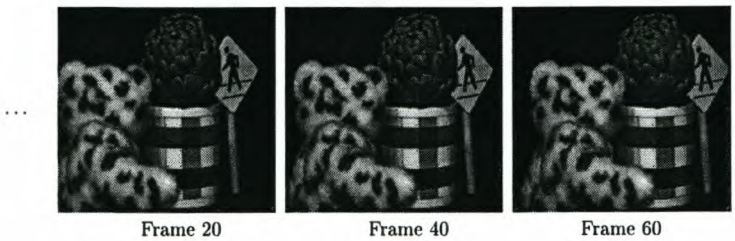


Figure 5.16: *A sample of frames from the artichoke sequence.*

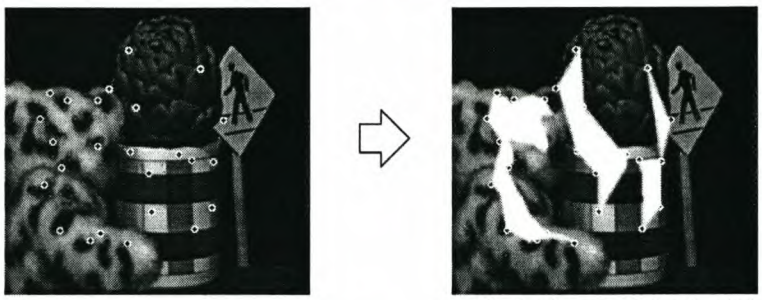


Figure 5.17: *An illustration of the hand-constructed mesh, showing its relation to the first frame.*

Figure 5.16 shows a couple of samples from the sequence, illustrating the left to right panning of the camera, as well as the scene itself.

5.4.3 Results and Conclusion

Since we have no ground truth data, the results are open to interpretation:

Figure 5.17 shows one processed frame, with the tracked features highlighted on the image. The other image shows the hand-constructed mesh which connects these points. The β values for these mesh-points were then estimated using the algorithm, and used to give the mesh shape. This mesh was then rotated around its centre, and rendered to produce figure 5.18.

The soft toy's back and tail can clearly be seen to extend from the head. It seems as if part of the ear is also well estimated. Also consider the sides of the cylinder, which are clearly vertical, and the artichoke which bulges out distinctly.

It is the author's subjective conclusion that this scene is handled well by the algorithm.

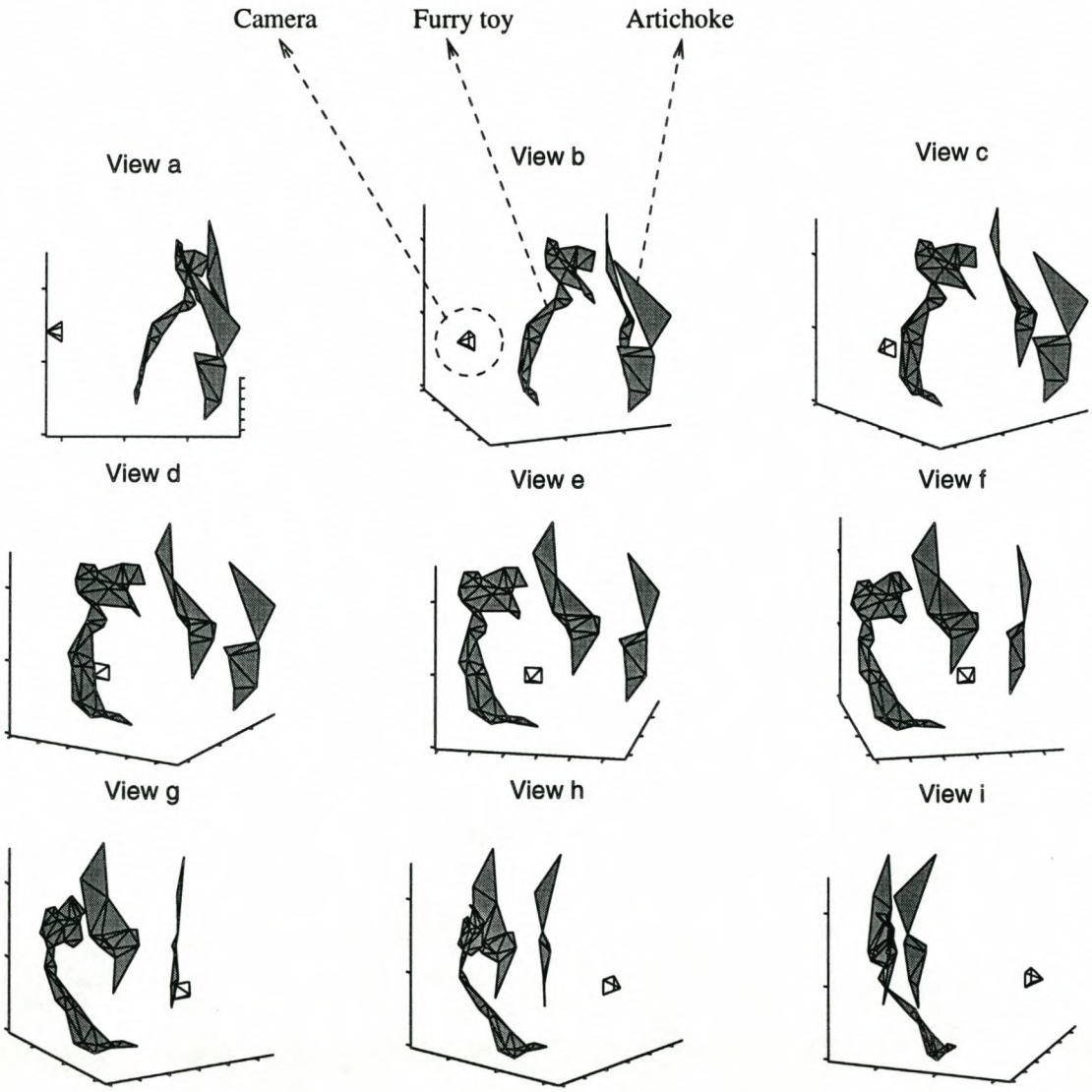


Figure 5.18: *A rotated rendering of the final structure estimation, as applied to the mesh.*

Chapter 6

Conclusion

In structure from motion the goal is to extract 3d information about a camera's movement relative to a scene, as well as the 3d structure of the scene, by using the 2d data contained in the video sequence produced by the camera.

There are a number of different approaches to this problem. Some attempt to only extract the camera's movement through the scene, leaving the calculation of the scene's structure as a kind of post-processing. Others simplify the camera model so that the problem becomes linear. Another approach utilises the Kalman filter, calculating both the perspective projection camera's motion as well as the structure of the scene. It relies on the Kalman filter's ability to extract the required information from the measurements, given the observation and system-update models.

In this thesis, an algorithm is proposed which implements the framework of the latter research, but utilises a new nonlinear Kalman filter, called the Unscented Kalman filter. Borrowing heavily from previous work, it implements the estimate of the camera and scene as two state variables, implemented in a dual-estimation filter.

The algorithm is compared against an implementation which uses a single Extended Kalman filter. Using various other types of data, including real-world footage with known ground-truths, it is shown that the UKF estimates the structure and motion very well, even in situations where the EKF-based algorithm fails to produce a result. It was shown that the UKF is particularly effective in situations where no prior knowledge of the system is available.

The present algorithm has the following constraints:

1. The sequence may contain only one moving object.
2. The object must be rigid.
3. Only features tracked through the entire sequence can be used.
4. Each iteration depends on the initial measurements.

The first two items allow only one rigid object in each data set. If we are able to describe a deformable object as a collection of smaller rigid objects, we may be able to use the algorithm unmodified, provided that we group each object's data into a separate data set. This, however, requires that we are able to recognise the multiple rigid objects as separate entities in the sequence.

In [10] Costeira develops a method of "object segmentation". Using information extracted from the output of the SVD batch algorithm [34], he is able to identify not only the number of objects in a scene, but also to group the measurements belonging to different objects into different data sets. However, the SVD algorithm assumes an orthographic projection camera, which requires a better understanding of Costeira's algorithm before it can be applied to the perspective projection problem.

An additional advantage which may result from the adaptation of Costeira's work, is the "segmentation" of a deformable object into a group of smaller rigid objects. For example, a deformable human hand may be segmented into a number of rigid digits. This will allow the second constraint to be partially removed, since it is not applicable to fully deformable/elastic objects.

Feature tracking is crucial in SfM, for without accurate feature tracking, our algorithm would be worthless. At the moment, however, there is a one-way flow of information from the feature tracker to the SfM algorithm. No information about the estimated image-plane translation of the features gleaned by the SfM algorithm is channelled back to the feature tracker, which may be used to improve the tracking ability of the feature tracker.

One obvious advantage includes limiting the search space for each feature in the next image using the *a priori* measurement estimate produced by the filter. Another less obvious advantage is the use of the estimated error covariance matrix to widen or narrow the search space.

Incorporating previously obscured features may also become easier. If information taken from the SfM output can be added to the information stored in the feature tracker, a newly acquired feature may be correctly "recognised" as a previously tracked one, which can lead to the re-initialisation of the feature using its last known estimate. Combined with an enhanced structure model (discussed below), this may completely remove the third and fourth constraints.

The third constraint restricts us to sets of data in which we are able to track all the points from the first frame through to the last frame in the sequence. This is caused by the structure model used by the algorithm, which requires the use of the initial measurements during each reconstruction of the object. New points cannot be introduced since their measurements in the initial frame are unknowable. The structure model is also the source of the last constraint.

In fact, the biggest design trade-off made in the development of the proposed algorithm

is the structure model. Currently, it records the initial measurements, and uses these measurements along with an extra state variable per feature to reconstruct the object. Any initial measurement error is therefore propagated in time, and prohibits the algorithm from ever producing a perfect reconstruction. This is reflected in the e_d scores from the experiment results in chapter 5.

However, this structure model reduces the structure estimate from the estimation of a time-varying variable to the estimation of a time-invariant variable, which also considerably simplifies the time-propagation equation for the structure model.

The enhancement possibility here is the design of a structure model which does not depend on the initial measurement, but instead uses the latest measurements in the reconstruction. This will force the *a priori* structure estimate equation to become more complex, and may also require a redefinition of the translation and rotation state variables.

This is acceptable, however, since it may allow newly detected features to be incorporated at any point in time. This in turn will allow "lost" features to be discarded more easily, since they can be replaced by these new "fresh" features.

To conclude, a better structure model may remove constraints three and four from the list above. This is entirely feasible and should be explored.

Bibliography

[1] "Portable graymap file format." http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/PGM.txt.

[2] "Povray, a free ray-tracing engine." http://www.povray.org.

[3] "Parametric Shape-from-Shading by Radial Basis Functions." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, apr 1997, Vol. 19, No. 4.

[4] BIRCHFIELD, S., "An implementation of the Kanade-Lucas-Tomasi feature tracker." http://www.vision.stanford.edu/~birch/klt. October 1998.

[5] BLOSTEIN, S. D. and CHANN, R. M., "The Use of Image Plane Velocity Measurements in Recursive 3-D Motion Estimation from a Monocular Image Sequence." *Canadian Conference on Electrical and Computer Engineering*, 1994, pp. 797–801.

[6] BOBICK, N., "Rotating Objects Using Quaternions." *Game Developer Magazine*, February 1998.

[7] BROIDA, T. J., CHANDRASHEKHAR, S., and CHELLAPPA, R., "Recursive 3-D Motion Estimation from a Monocular Image Sequence." *Trans. Aero. and Elec. Sys.*, July 1990, Vol. 26, No. 4, pp. 639–656.

[8] BROOKS, M., CHOJNACKI, W., and VAN DEN HENGEL, A., "Solving the Shape from Shading Problem on the CM5." *IEEE Conference on Computer Architectures for Machine Perception*, 1995.

[9] CHEN, C. K. C. G., *Kalman filtering with Real-Time Applications*. New York: Springer-Verlag, 1987.

[10] COSTEIRA, J. P. S. A., *A multi-body factorization method for motion analysis*. PhD thesis, Universidade Técnica de Lisboa, Lisbao, 1995.

[11] FAUGERAS, O., *Three-Dimensional Computer Vision*. Isbn 0-262-06158-9 edition. Cambridge, Massachusets, London, England: The MIT Press, 1993.

[12] FAUGERAS, O., *Three-Dimensional Computer Vision*, p. 15. Cambridge, Massachusets, London, England: The MIT Press, Isbn 0-262-06158-9 edition, 1993.

[13] FAUGERAS, O., *Three-Dimensional Computer Vision*, pp. 144–146. Cambridge, Massachusets, London, England: The MIT Press, Isbn 0-262-06158-9 edition, 1993.

[14] FORSYTH, D. A., IOFFE, S., and HADDON, J., "Bayesian Structure from Motion." *Int. Conf. on Computer Vision (ICCV)*, 1999, pp. 660–665.

[15] HAYKIN, S., *Adaptive filter theory*, pp. 302–337. Upper Saddle River, New Jersey: Prentice Hall, Inc., 1996.

[16] JEBARA, T., AZARBAYEJANI, A., and PENTLAND, A., "3D Structure from 2D Motion." *IEEE Signal Processing Magazine, "3D And Stereoscopic Visual Communication"*, May 1999, Vol. 16, No. 3.

[17] JULIER, S. J., "The Scaled Unscented Transformation." *To appear in Automatica*, feb 2000.

[18] JULIER, S. J. and UHLMANN, J. K., "A General Method for Approximating Nonlinear Transformations of Probability Distributions." tech. rep., Robotics Research Group, Dept. of Engineering Science, University of Oxford, November 1996.

[19] JULIER, S. J. and UHLMANN, J. K., "A New Extension of the Kalman Filter to Nonlinear Systems." *A new extension of the Kalman filter to nonlinear systems. In Int. Symp. Aerospace/Defense Sensing, Simul. and Controls, Orlando, FL*, 1997.

[20] JULIER, S. J., UHLMANN, J. K., and DURRANT-WHYTE, H. F., "A New Approach for Filtering Nonlinear Systems." *Proceedings of the 1995 American Control Conference*, 1995, pp. 1628–1632.

[21] LAMPORT, L., LaTeX—*A Document Preparation System*. Reading, Massachusetts: Addison-Wesley, 1994.

[22] LASENBY, J., LASENBY, A. N., DORAN, C. J. L., and FITZGERALD, W. J., "New geometric methods for Computer Vision: an application to structure and motion estimation." *International Journal of Computer Vision*, 1998, Vol. 26(3), pp. 191–213.

[23] LUONG, Q. and FAUGERAS, O., "The Fundamental Matrix: Theory, Algorithms, and Stability Analysis." *International Journal of Computer Vision*, 1995.

[24] LUONG, Q. T., DERICHE, R., FAUGERAS, O., and PAPADOPOULO, T., "On determining the fundamental matrix: analysis of different methods and experimental results." Tech. Rep. RR-1894, INRIA, Sophia Antipolis, France, 1993.

[25] MAYBECK, P. S., *Stochastic models, estimation, and control, Volume 1*. ACADEMIC PRESS, 1979.

[26] MORITA, T. and KANADE, T., "A Sequential Factorization Method for Recovering Shape and Motion from Image Streams." *Proceedings of 1994 ARPA Image Understanding Workshop*, November 1994, Vol. II, pp. 1177–1188.

[27] O'KENNEDY, B. Master's thesis, Department of Electrical and Electronic Engineering, University of Stellenbosch, Western Cape, South Africa, February 2002.

[28] PERVIN, E. and WEBB, J. A., "Quaternions for Computer Vision and Robotics." *Computer Vision and Pattern Recognition*, 1983, pp. 382–383.

[29] PINKER, S., *How the mind works*. Penguin Books Ltd, 27 Wrights Lane, London W8 5TZ, England: The Penguin Group, 1997.

[30] POELMAN, C. J. and KANADE, T., "A Paraperspective Facrotization Method for Shape and Motion Recovery." Tech. Rep. CMU-CS-93-219, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, December 1993.

[31] SAMARAS, D., METAXAS, D., FUA, D., and LECLERC, P., "Variable Albedo Surface Reconstruction from Stereo and Shape from Shading." *Submitted to the IEEE Computer Vision and Pattern Recognition Conference*, 2000.

[32] SORENSON, H. W. (Ed.), *Kalman Filtering: Theory and Application*. New York: IEEE Press, 1985.

[33] STEWART, A. J. and LANGER, M. S., "Towards Accurate Recovery of Shape from Shading under Diffuse Lighting." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, September 1997, Vol. 19, No. 9, pp. 1020–1025.

[34] TOMASI, C. and KANADE, T., "Shape and motion from image streams: A factorization method - full report on the orthographic case." Tech. Rep. CMU-CS-92-104, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1992.

[35] TRUCCO, E. and VERRI, A., *Introductory Techniques for 3-D Computer Vision*. Englewood Cliffs, NJ: Prentice Hall, 1998.

[36] TSAI, R., "A Versatile Camera Calibration Technique for High Accuracy 3D Machine Vision Metrology Using Off the Shelf TV Cameras and Lenses." Tech. Rep. IBM Research Report, RC 11413, 1985.

[37] WAN, E. A. and VAN DER MERWE, R., "The Unscented Kalman Filter for Nonlinear Estimation." *Proc. of IEEE Symposium 2000 (AS-SPCC)*, October 2000.

[38] WAN, E. A., VAN DER MERWE, R., and NELSON, A. T., "Dual Estimation and the Unscented Transformation." *Advances in Neural Information Processing Systems 12*, 2000, pp. 666–672.

[39] WELCH, G. and BISHOP, G., "An Introduction to the Kalman Filter." `http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html`. December 1997.

[40] WHEELER, M. D. and IKEUCHI, K., "Iterative Estimation of Rotation and Translation using the Quaternion." Tech. Rep. CMU-CS-95-215, Computer Science Department, Carnegie Mellon University, December 1995.

[41] YIU CHANN, R. M., "Recursive estimation of 3-D motion and structure in image sequences based on measurement transformations." Master's thesis, Queen's University, Kingston, Ontario, Canada, September 1994.

[42] ZHANG, Z., "Determining the epipolar geometry and its uncertainty: A review." Tech. Rep. RR-2927, INRIA, Sophia Antipolis, France, 1996.

[43] ZHAO, L., "Recursive estimation of 3-D Motion Trajectories from image sequences using measurements of feature point positions and optical flow." tech. rep., Queen's University, Kingston, Ontario, Canada, May 1998.

Appendix A

Code

This appendix discusses the data and source code directory structure with which the proposed algorithm is implemented. System specifications of the development machines are given first, followed by an overview of the source directory tree. The last section is a user's guide for various programs.

A.1 Hard- and software specifications

The author's system, with the following specifications, was used for feature tracking and write-up:

Computer	**orb.dsp.sun.ac.za**
Processor	Intel Celeron 300A
Operating system	Redhat Linux 7.2
Kernel version	2.4.7-10
C/C++ Compiler	gcc version 2.96 20000731
python	1.5.2
ImageMagick	5.3.8
povray	3.1g.Linux.gcc
fig2dev	3.2 Patchlevel 3d

A remote machine with the following specifications was used for the Matlab execution:

Computer	**percy.dsp.sun.ac.za**
Processor	AMD 700
Operating system	Red Hat Linux release 6.2 (Zoot)
Kernel version	2.2.14-5.0
Matlab version	5.3.0.10183 (R11)

A.2 The directory tree: `thesis`

The top-level directory branches into the following subdirectories:

- thesis
  - bin
    - klt_track
    - script
    - bin
  - data
    - artichoke
    - cube
    - radio
    - output
  - matlab
    - chann
    - cube
    - flip
    - general
    - jam
    - libmat
    - povray
  - papers
    - fig
    - lib
    - thesis
    - output

Each directory has a `Makefile`, which contains commands which can be executed to build binaries, process files, or clean directories, as required. To see the commands that each Makefile provides, type `make` at the prompt. For example, typing this in the `thesis/matlab` directory produces the output:

```
[chris@orb matlab]$ make
##
## clean        Runs 'make clean' in all subdirectories
##
```

Therefore, to clean up the `thesis/matlab` directory, simply type `make clean`.

A.2.1 `thesis/bin`

The `thesis/bin` directory contains all the C/C++ source code used in the thesis, as well as scripts written in Python. The `Makefile` can be used to compile the various binary executables and place them in the `thesis/bin/bin` directory, so that other programs can easily find them.

`thesis/bin/klt_track`

Feature tracking software based on the Kanade-Lucas-Tomasi feature-detection and tracking algorithm. The code was written as a library of functions by Stan Birchfield and can be downloaded from [4]. The version used in this document has been superficially altered and used in the `stereo_track.c` program to enable feature tracking in a stereo-camera setup.

`thesis/bin/script`

The main Python script directory. The scripts are use for various automated file-manipulation duties, such as converting from one image format to another.
`batch_convert` is a batch file-conversion script. It takes three arguments: an input pattern, an output pattern, and optional arguments for the `convert` tool.
`batch_convert.py` does the same as `batch_convert`, although in this case it accepts a file list, instead of simply converting all the matching files in the directory.
`batch_rename` renames all the files containing a specific pattern so that the pattern is placed with another given pattern.
Each of these scripts produce usage instructions when executed with no arguments

`thesis/bin/bin`

This is the main binary executable directory. All of the executables (except for the scripts in `thesis/bin/scripts`) are copied into this directory after compilation, so that they are easily locate-able by the other scripts and executables.

A.2.2 thesis/data

The `data` directory contains all the test data for the algorithm. These include all the Povray scripts as well as source images in various image file formats.

`thesis/data/artichoke`

This directory contains source images and corresponding feature lists for the experiment in section 5.4.
`img*.png`: Source images.
`feat*.mat`: Final feature lists of tracked features.

`thesis/data/cube`

The synthetic povray sequence of the spinning cube in experiment 5.2 was generated using the file `cube_0.pov` found in this directory. The `Makefile` contains the commands

required to generate the images and process them with the feature tracking software.
`cube_0.pov`: Povray script.
`my.ini`: Povray timer settings for animation.

`thesis/data/radio`

This directory contains the input data and calculated ground truth for the radio experiment from section 5.3.
`dual_left*.txt`: Lists of the tracked feature points in the primary (left) camera over time.
`dual_right*.txt`: Records of the tracked features between the primary and secondary (right) camera per frame.
`dual_valid*.txt`: Records of the features that are track-able in both cameras up to every point in time.
`feat*.mat`: Output of the feature tracker.
`data3d-*.txt`: Ground-truth for track-able points in left and right camera. Calculated with O'Kennedy's software [27].
`mono*.txt`: The final output of the feature tracker, which contains the lists of features track-able through the left image.

`thesis/data/output`

Once the images have been processed with feature tracking software, the feature lists are copied into this directory, so that Matlab scripts can easily locate and process them.

A.2.3 `thesis/matlab`

The main Matlab directory where the proposed algorithm's scripts reside.

`thesis/matlab/general`

This directory contains scripts for the code shared between different SfM algorithm implementations. Examples include the error generating code, plotting functions and a Povray-script generator.

`thesis/matlab/libmat`

Matlab code library for quaternion mathematics and Unscented Kalman filter code.

`thesis/matlab/flip`

The main implementation of the proposed algorithm. This code accepts input information and processes it according to algorithm developed in this thesis. User documentation for this code can be found in section B.

`thesis/matlab/chann`

An implementation the SfM algorithm by Broida et al, based on the EKF. It was written by Blostein and Chann [5] and kindly provided by Lin Zhao. It has been superficially modified to enable comparison with the algorithm proposed in this work.

`thesis/matlab/jam`

Another implementation of the proposed algorithm, which is modified to accept the synthetic data used in the EKF algorithm mentioned above. It also allows a Monte-Carlo analysis of performance, in order to allow comparison between it and the above-mentioned algorithm.

`thesis/matlab/cube`

A Matlab script which computes the state of the synthetic cube from the experiment in section 5.2. It uses the same setup as provided to Povray, in order to calculate the ground-truth data required for the verification of the output of the proposed algorithm.

`thesis/matlab/povray`

Povray scripts representing the estimated structure and motion using spheres are written into this directory with the `create_povray.m` script.

A.2.4 `thesis/papers`

This document's source resides in the **`thesis/papers/thesis`** directory. It will not be discussed in detail, since it is a standard LaTeX [21] document. The Encapsulated Postscript graphics used in the document are found in **`thesis/papers/fig`**, and the style files and Bibliography files are found in **`thesis/papers/lib`**.

Appendix B

Software usage

This chapter shows how to run the implemented algorithm to either reproduce the experiments in chapter 5, or to process a new sequence of images.

B.1 Compiling `stereo_track`

The feature tracking program, `stereo_track`, is compiled by executing

```
$ cd thesis/bin/klt_track
$ make output
```

at the prompt. For general usage instructions, type

```
$ ./stereo_track
```

in the same directory.

B.2 Input file type

The images must be in Portable grey-map format [1], with filenames of the form `[base]-[xxx].pgm` where `base` is a shared pref-fix and `xxx` is the three-digit number of the frame in the sequence. Therefore `car-001.pgm` is a valid filename, which is frame one in the `car` sequence.

For example, to process an image sequence with the filenames:

`car-001.pgm, car-002.pgm, car-003.pgm, ... car-020.pgm`

enter the following command at the prompt:

```
$ stereo_track car _ - 1 20
```

This will produce the output files

`feat001.mat, feat002.mat, feat003.mat, ... feat020.mat`

in the same directory. These files must now be copied to the `thesis/data/output` directory to enable the Matlab programs to find and use them:

```
$ cp feat*.mat [path to thesis/output]
```

The images in the `thesis/data` directory are already in the PGM format. The accompanying `Makefile` in each directory is set up so that

```
$ make output
```

uses the pre-compiled `stereo_track` program to track the features in the image sequences, and copy the output feature lists into the `thesis/data/output` directory automatically.

B.3 Running an experiment

Once the images in the sequence reside in the `thesis/data/output` directory, and they have been processed with the `stereo_track` program, there should be a collection of `feat[xxx].mat` files in the same directory.

Start Matlab, and change into the `thesis/matlab/flip` directory. Once there, type the command

```
>> main
```

The software should now start, by displaying the information

```
filter: UKF
Using data from [klt_track] with images of 640x480
[68] number of features per frame for [98] frames
motion[5.0e-07, 5.0e-06, 1.6e-05]
shape [5.0e-08, 5.0e-07, 1.6e-05]
#######...
```

The software should terminate with a result such as

```
rms 2d error:       0.0037134388
rms scaling error: 0.0066217175
>>
```

where the e_d value is given by the `rms 2d error` value, and the e_s value is given by the `rms scaling error` value (if the 3d ground truth is available).

B.4 Running a comparison between the EKF- and the UKF-based algorithms

To run one of the comparisons from section 5.1, another implementation of the proposed algorithm is used. This implementation has been hard-coded to generate and use the same data as the EKF-based algorithm implemented by Chann and Blostein [5], and supplied by Lin Zhao.

Additionally, the code has been modified to enable it to do a Monte-Carlo run. To enable this, edit the file `thesis/matlab/jam/mc_init.m` and set the parameter `mc_runs` to the number of required runs.
Edit the file `thesis/matlab/jam/chann_initialize.m` to set up the type of experiment. The `total_frames` parameter sets the number of frames in each run. The `movement`, `accelerate` and `abrupt_change` parameters control the type of experiment. Choose one from the list by un-commenting it.
The `percent_error` parameter determines how much error the initial information supplied to the algorithm contains. A value of -1 means that no initial information is supplied.
Start Matlab, and change to the `thesis/matlab/jam` directory. In order to run the proposed UKF-based algorithm, type

```
>> main
```

at the prompt. In order to run the EKF-based algorithm, type

```
>> chann_ekf
```

Comparison of the two filters can be done from the error values produced, as was done in section 5.1

B.5 Running the radio experiment

The experiment from section 5.3 requires special treatment, since the ground truth must be loaded from the files as well, in order for the errors to be calculated. The measurement data must also be undistorted, according to the camera-calibration data supplied by Brian O'Kennedy using his calibration techniques [27].
Type the command

```
$ make output
```

in the `thesis/data/radio` directory. This will copy the `mono[xxx].txt` files into the `thesis/data/output` directory.
Edit the `thesis/matlab/flip/initialise.m` file, and uncomment the `input_data='mono';` line.
Start Matlab, and type the command

```
>> main
```

at the prompt.

B.6 Displaying the results

Several different results can be displayed, using the different plotting routines:

- `plot_2d`: Plots the 2d image plane coordinates over time, contained in the ground-truth data, the actual measurements, and the estimated measurements.
- `plot_3d`: Plots the 3d CCS ground-truth as well as the estimated 3d CCS information over time.
- `plot_shape`: Displays the final OCS object reconstruction, as well as the ground truth.
- `plot_trans`: Plots the estimated as well as ground truth translation data over time.
- `plot_rot`: Plots the estimated as well as ground truth rotation data in terms of the quaternion values over time.

The ground truth will only be plotted if available, depending on the experiment.